

NEURO-SYMBOLIC LEARNING OF ANSWER SET PROGRAMS
FROM RAW DATA

Daniel Thomas Cunningham

Department of Computing
Imperial College London

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

June 2025

Statement of Originality

I, Daniel Cunningham, declare that the work in this thesis is my own. The work of others has been appropriately referenced. A full list of references is given in the bibliography.

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Licence (CC BY NC-SA). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that; you credit the author, do not use it for commercial purposes and share any derivative works under the same licence. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Abstract

Artificial Intelligence (AI) is becoming increasingly integrated in society. This raises serious questions of the robustness of the technology. Neuro-Symbolic AI is well-suited to help answer these questions. Many existing neuro-symbolic systems focus on training a neural network given manually engineered background knowledge. However, this can be laborious, and the complete knowledge may not be available. The question is, *can we leverage recent advancements in the robustness and expressivity of symbolic machine learning to learn knowledge from raw data?*

To integrate both neural and symbolic learners, there are various challenges to address: (1) Can exact symbolic knowledge be learned from noisy neural network predictions? (2) Can symbolic knowledge be learned in an end-to-end fashion, since symbolic learners are not differentiable? (3) Can a neuro-symbolic learning framework scale to complex tasks? In this thesis, we propose three novel approaches, capable of learning highly expressive knowledge in the language of Answer Set Programming, defined in terms of primitive concepts extracted from raw data.

First, we investigate the robustness of two state-of-the-art symbolic learners when learning complex knowledge, given noisy concept predictions from pre-trained neural networks. Empirical results demonstrate that symbolic knowledge can be learned accurately, even when large percentages of training data are subject to distributional shifts, which causes the networks to predict incorrectly with high confidence. Second, we develop an end-to-end architecture that iteratively trains both neural and symbolic components. We demonstrate that our approach outperforms a variety of baselines, achieving state-of-the-art results. Finally, we explore scalable solutions for neuro-symbolic learning, and develop an architecture that leverages the implicit knowledge embedded within large vision-language foundation models, to extract primitive concepts from raw data. Our evaluation demonstrates the scalability and data efficiency gained with respect to state-of-the-art neuro-symbolic AI methods, in cases where the symbolic knowledge is both given, and learned.

To my family, Lesley, Robert, and Aidan.

Acknowledgements

There is one name on the front, but there was a team effort behind the scenes. Every PhD is a journey, and this is no exception. There are many people to thank, without whom, I wouldn't have had a chance of completing such an intense program of work.

Firstly, the biggest thanks deservedly goes to my advisors, Prof. Alessandra Russo, Prof. Jorge Lobo, and Dr. Mark Law. Alessandra, our group lead, has unlimited energy, passion, and enthusiasm for her work. This also applies when supporting her students, often in the early hours of the morning. Alessandra always finds the time to lead, guide, and mentor. One could not ask for a better advisor. Jorge has a critical eye for detail, and has suggested many items of work which have made this thesis. Thank you Jorge for all your efforts, especially when you moved back to Barcelona half-way through my studies. Mark has been an invaluable asset throughout my PhD. Mark's ASP knowledge is world-class, and he has helped me define learning tasks, optimise ILASP and FastLAS for my use-cases, and suggest new directions of work. Thank you Mark. Thanks must also go to Dr. Krysia Broda, for advice throughout my PhD programme, but also for assessing me in my early and late stage reviews.

Thank you to my former employers, the wonderful people in the IBM UK Research team. I would never have had the opportunity to pursue a PhD, if it wasn't for these folks. In particular, thank you to Dr. Geeth de Mel for making the connection and introducing me to Alessandra. Dr. Dave Braines, for demonstrating that pursuing a PhD alongside work was even remotely possible. Helen Stanton, one of the best managers around, for all your support and guidance. Graham White, for introducing me to machine learning with your department challenge competitions. Dr. Sue Johnson, and Dr. Peter Waggett for signing it all off, and giving me a substantial amount of time "on the clock" to work on my PhD. Thanks to all of my former colleagues for the love and support over the years. I started my studies in the midst of an international research programme at IBM, called DAIS-ITA. Thanks to the DAIS-ITA funding partners Dstl, and US ARL, for funding some of the early work that led to this thesis, especially the work presented in Chapter 3.

Thank you to my immediate, and extended family. In particular, thanks to my parents, Lesley and Robert, and my brother Aidan. You have always nurtured me and encouraged me to pursue opportunities to develop myself. I am forever grateful for all of your support, especially during the challenging moments. A lot of credit must also go Siobhan, for supporting me day-to-day, and tolerating many hours of me working, including evenings, weekends, and "holidays". To all my friends, whether at home in Newcastle, from my undergraduate degree in Sheffield, or in Winchester, where I lived for the majority of this PhD programme, thank you for all of your support and encouragement. A PhD without having fun outside of work would be very challenging. I am extremely grateful for all of the running, and socials over the years. Thanks must also go to my fellow pigeons of the SPIKE research group, who accepted me as a part-time student who occasionally showed up on a Friday. All of you inspired me a great deal with your knowledge, work ethic, and intelligence. In particular, it was great to study alongside Yaniv Aspis. We started at the same time in 2020, and now are concluding our studies at the same time also.

Finally, thanks to my current employers, Durable, for giving me an opportunity to continue working within neuro-symbolic AI, and for understanding my commitments in writing up this thesis, whilst I started a new job and settled in to the post-PhD chapter in the USA.

Contents

List of Figures	xiii
List of Tables	xix
Acronyms	xxi
Notation	xxiii
1 Introduction	1
1.1 Motivations	2
1.2 Technical Challenges	3
1.3 Contributions	4
1.4 Publications	7
1.5 Thesis Structure	7
2 Background	9
2.1 Deep Learning	9
2.1.1 Artificial Neural Networks	9
2.1.2 Advanced Neural Network Architectures	14
2.1.3 Foundation Models	20
2.1.4 Uncertainty	21
2.2 Symbolic AI	24
2.2.1 Computational Logic	25
2.2.2 Answer Set Programming	30
2.2.3 Learning from Answer Sets	32
3 Feed-Forward Neuro-Symbolic Learning	35
3.1 Problem Formulation	37
3.2 FF-NSL Framework	38
3.2.1 Extracting Primitive Concepts from Raw Image Data	39
3.2.2 Learning Symbolic Knowledge	40
3.2.3 Inference	41
3.3 Neuro-Symbolic Evaluation Tasks	42
3.4 RQ1: Evaluating the Learned Hypotheses	51
3.4.1 Methodology	51
3.4.2 Experimental Setup	52
3.4.3 Follow Suit Winner	54
3.4.4 Sudoku Grid Validity	61

3.4.5	Real-world Datasets	66
3.5	RQ2: Evaluating the FF-NSL Framework	69
3.5.1	Follow Suit Winner	69
3.5.2	Sudoku Grid Validity	70
3.5.3	Real-world Datasets	71
3.6	RQ3: Evaluating the FF-NSL Framework with Uncertainty	73
3.7	Summary	75
4	Iterative Neuro-Symbolic Learning	77
4.1	Problem Formulation	78
4.2	NSIL Framework	79
4.2.1	Symbolic Component	79
4.2.2	Neural Component	80
4.2.3	Corrective Examples	82
4.3	Evaluation	86
4.3.1	Experimental Setup	87
4.3.2	Cumulative Arithmetic	88
4.3.3	Two-Digit Arithmetic	90
4.3.4	The Hitting Set Problem	98
4.3.5	Investigating the λ Hyper-Parameter	103
4.4	Challenges	105
4.4.1	Trying to Prove Convergence	105
4.4.2	NeurASP	107
4.5	Summary	109
5	Exploring Decoupled Neuro-Symbolic Learning and Reasoning	111
5.1	Image Clustering	112
5.2	Labelling the Clusters	117
5.2.1	Fully Differentiable Cluster Labelling	118
5.2.2	Breadth-First Search	126
5.3	Summary	130
6	Scalable Neuro-Symbolic Learning and Reasoning with Foundation Models	131
6.1	Problem Formulation	134
6.2	NeSyGPT Framework	135
6.2.1	BLIP Fine-Tuning	135
6.2.2	Learning a Hypothesis	137
6.3	Evaluation	140
6.3.1	Experimental Setup	140
6.3.2	Two-Digit Arithmetic	143
6.3.3	Follow Suit Winner	146
6.3.4	Plant Hitting Sets	149
6.3.5	CLEVR-Hans	154
6.3.6	Reducing the Manual Engineering using a LLM	155
6.3.7	Zero-Shot Evaluation	161
6.4	Summary	163

7	Related Work	165
7.1	Directly Related Work	165
7.1.1	Hybrid Approaches	165
7.1.2	Learning Knowledge in a Differentiable Manner	167
7.2	Other Related Work	170
7.2.1	Training Neural Networks to Simulate Logical Formulae	170
7.2.2	Training Neural Networks with Logic Programs	171
7.2.3	Training Neural Networks with Differentiable Knowledge	173
7.2.4	Symbolic Reasoning from Pre-Trained Neural Networks	173
7.2.5	Leveraging Foundation Models	174
7.2.6	Summary	176
8	Conclusion	177
8.1	Summary of Contributions	177
8.2	Future Work	179
	Bibliography	181
A	LAS Task Specifications	195
B	NeSyGPT Example Generators	201

List of Figures

2.1	Example CNN architecture with three convolution layers, when tasked with classifying an image as “Snow”, “Jungle”, or “Desert”.	14
2.2	Visual overview of the information flow within an LSTM cell.	18
2.3	Popular t -norms that model conjunction in fuzzy logic.	29
3.1	Example inputs $\mathbf{x}_i$, and associated meta-data m_i for the Follow Suit Winner task. The meta-data indicates which player played each card. In this example, the label is $y = 3$	38
3.2	Example $\mathbf{x}_i$ and m_i inputs for the Follow Suit Winner task with a distributional shift applied. In this case, standard card images are substituted with images from an alternative deck.	38
3.3	General FF-NSL learning framework, shown for a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$	39
3.4	Inference for a single data point $\langle X, M \rangle$	42
3.5	FF-NSL learning framework, instantiated for a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ in the Follow Suit Winner task.	43
3.7	Neural network performance under distributional shifts in the Follow Suit Winner task. Each row presents the accuracy of a neural network on a particular card deck. The confidence score distribution reflects the percentage of neural network predictions within the varying confidence intervals. For example, in the Softmax Captain America case, 96% of the neural network predictions were made with confidence in the interval $[0.95, 1]$, despite the network only achieving 0.0697 accuracy.	45
3.6	Example playing card images used in the Follow Suit Winner task.	45
3.8	Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Sudoku Grid Validity 4×4 task. .	46
3.9	Neural network performance under distributional shifts, Sudoku Grid Validity task. Each row presents the accuracy of a neural network with and without the distributional shift (rotation) applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval. For example, in the 4×4 Softmax rotated case, 76% of neural network predictions are made with confidence in the interval $[0.95, 1]$, despite the network only achieving 0.137 accuracy.	47
3.10	Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Crop Yield Prediction task. . . .	48

3.11	Neural network performance under distributional shifts. Crop Yield Prediction task. Each row presents the accuracy of a neural network with and without the distributional shift (hue filter) applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval. For example, in the Softmax hue shift case, 48% of neural network predictions are made with confidence in the interval $[0.95, 1]$, despite the network only achieving 0.0765 accuracy.	49
3.12	Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Indoor Scene Classification task. .	50
3.13	Neural network performance under distributional shifts. Indoor Scene Classification task. Each row represents the accuracy of a neural network with and without the distributional shift applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval. . .	50
3.14	Accuracy of the learned hypotheses when increasing percentages of training data points are subject to distributional shifts, Follow Suit Winner task.	54
3.15	FF-NSL Softmax vs. FF-NSL EDL-GEN. Accuracy of the learned hypotheses with 95–100% distributional shifts, using 50 repeats. Follow Suit Winner task. .	55
3.16	The effect of distributional shifts on the percentage of consistent WCDPI examples generated. Follow Suit Winner task.	56
3.17	The percentage of experimental repeats that learned the correct follow suit rules, when 95% of input data points were subject to distributional shifts.	57
3.18	Distribution of playing card rank predictions for Softmax and EDL-GEN neural networks when the Captain America and Adversarial Captain America decks are used to apply distributional shift for 95% of input data points.	58
3.19	WCDPI example weight penalty ratio, 95–100% shifts. Follow Suit Winner task.	59
3.20	Interpretability of the learned hypotheses, Follow Suit Winner task.	60
3.21	Learning time. Follow Suit Winner task.	61
3.22	Learned hypothesis accuracy with increasing percentages of data points subject to distributional shifts, Sudoku Grid Validity task.	62
3.23	FF-NSL Softmax vs. FF-NSL EDL-GEN. Average accuracy of learned hypotheses over 50 repeats. Sudoku Grid Validity task.	63
3.24	WCDPI example weight penalty ratio, Sudoku Grid Validity task.	63
3.25	The effect of applying distributional shifts on the percentage of consistent WCDPI examples, Sudoku Grid Validity task.	64
3.26	Interpretability of the learned hypotheses, Sudoku Grid Validity task.	65
3.27	Hypothesis learning time, Sudoku Grid Validity task.	66
3.28	Learned hypothesis accuracy.	67
3.29	The effect of setting WCDPI example weight penalties based on neural network confidence scores, compared to constant penalties, Crop Yield Prediction task. 95-100% shifts, 50 repeats.	67
3.30	Interpretability of the learned hypotheses. Real world datasets.	68
3.31	Hypothesis learning time. Real world datasets.	69
3.32	Accuracy of the FF-NSL framework when training and test data were subject to the same % of distributional shifts. Follow Suit Winner.	70
3.33	Neural network accuracy for predicting playing cards when test data points were subject to distributional shifts. Follow Suit Winner task.	71
3.34	Accuracy of the FF-NSL framework when training and test data are subject to distributional shifts. Sudoku Grid Validity task.	72

3.35	Accuracy of the FF-NSL framework when training and test data were subject to distributional shifts. Real-world datasets.	72
3.36	Example data point from the Sudoku Grid Validity 4×4 task.	73
3.37	FF-NSL Framework Uncertainty Analysis, Sudoku Grid Validity 4×4 task. . . .	75
4.1	(a) NSIL learning with a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. (b) NSIL inference with a single input $\langle X, M \rangle$	80
4.2	Cumulative Addition corrective examples	89
4.3	Cumulative Arithmetic learned hypotheses	90
4.4	Cumulative Arithmetic network accuracy	90
4.5	E9P corrective examples	91
4.6	Two-Digit Arithmetic results with reduced training sets.	92
4.7	Two-Digit Arithmetic learned hypotheses	93
4.8	Two-Digit Addition accuracy breakdown.	93
4.9	Two-Digit E9P accuracy breakdown.	94
4.10	Corrective examples for $Z = \langle 3, 5 \rangle$, $M = \langle \rangle$, and $y = 8$, Addition 100% task, iteration 2.	94
4.12	Neural Network confusion matrices	95
4.11	Corrective examples for $Z = \langle 2, 3 \rangle$, $M = \langle \rangle$, and $y = 5$, Addition 10%, iteration 2.	95
4.13	Two-Digit Arithmetic accuracy comparison with no corrective examples. Average over 20 repeats.	97
4.14	Two-Digit Arithmetic learning time vs. accuracy with reducing training set sizes.	98
4.15	Hitting Set corrective example combination used during the initial ILASP learning task.	100
4.16	Hitting Set accuracy	101
4.17	Hitting Set learned hypotheses	101
4.18	Hitting Set accuracy comparison with no corrective examples. Average over 20 repeats.	103
4.19	Hitting Set learning time vs. accuracy.	104
4.20	Hypothesis accuracy with varying λ , E9P and HS tasks. Error bars indicate standard error.	105
4.21	NSIL performance in a restricted search space for the Two-Digit Addition task. (a) The accuracy of the learned hypothesis only. (b) The accuracy of the neural network, the learned hypothesis, and the overall (end-to-end) accuracy. This shows NSIL is not guaranteed to converge to an optimal hypothesis.	106
4.22	NSIL performance on the Two-Digit Addition task with CIFAR-10 images. . . .	107
4.23	Embed2Sym performance on the CIFAR-10 Addition task with an increasing number of digits. (Top) a CNN is used as the perception component. (Bottom) a ResNet56 is used. (Left) The accuracy of the perception component on classifying concept values. (Middle) The MAE of the fully neural architecture from stage 1 of Embed2Sym (perception component + MLP). (Right) The final neuro-symbolic task accuracy from Embed2Sym.	108
4.24	The embedding spaces of the perception components within Embed2Sym during training, as the number of digits increases.	109
5.1	Training a Variational Autoencoder on MNIST images.	114
5.2	Evaluating the SPICE clustering method on CIFAR-10 test images.	115

5.3	Playing card clustering with full card images.	115
5.4	Playing card clustering with Rank Suit Crop images.	116
5.5	Evaluating the WideResNet feature extractor on CIFAR-10 test images.	117
5.6	Fully differentiable end-to-end architecture for learning a cluster mapping function.	118
5.7	Fully differentiable end-to-end architecture for learning a cluster mapping function, where only the cluster mapping function is trainable.	119
5.8	Fully differentiable end-to-end architecture for learning a cluster mapping function, with all of our modifications added.	121
5.9	Test set accuracy results when training neural surrogate models to learn complex knowledge.	122
5.10	Expected vs. Actual matrices output from the trained mapping network on the Follow Suit Winner task.	125
5.11	Breadth-first search to label the clusters, shown with the MNIST Addition task. The green path shows the correct mapping.	127
6.1	Example reasoning shortcut in the autonomous driving domain. Source: (Maconato et al., 2024).	132
6.2	NeSyGPT architecture with one data point in the Follow Suit task. The goal is to learn the rules of the game: The winner is the player with the highest ranked card with the same suit as Player 1. (a) BLIP is fine-tuned using playing card images and natural language questions and answers. (b) A hypothesis is learned from BLIP predictions. Note in (a), fine-tuning occurs with both suit and rank answers.	136
6.4	Two-Digit Arithmetic task accuracy when the hypotheses are given.	144
6.3	Example data point for the BLIP Single Image baseline. in the case of Addition, BLIP is fine-tuned to output $y = 8$	144
6.5	Two-Digit Arithmetic task accuracy when the hypotheses are learned.	145
6.6	Two-Digit Arithmetic learned hypotheses	145
6.7	Embedding space comparison between the Embed2Sym variants on the Two-Digit Arithmetic tasks. (Left) Embed2sym*. (Right) Embed2Sym. (Top) Addition. (Bottom) E9P.	146
6.8	Example data point for the BLIP Single Image baseline. in this case, BLIP is fine-tuned to output $y = 3$	147
6.9	Follow Suit Winner task accuracy when the symbolic rules are given.	148
6.10	Follow Suit Winner task accuracy when the symbolic rules are learned.	149
6.11	Follow Suit Winner learned hypothesis	149
6.12	Embed2Sym embedding spaces — Follow Suit 4 Standard.	150
6.13	Plant Hitting Sets example data point. The meta-data M indicates which set each element belongs to. The label is $y = 1$ as there is a hitting set of size ≤ 2 with the elements Apple Black Rot, and Tomato Bacterial Spot.	150
6.14	Example data point for the BLIP Single Image baseline. In this case, BLIP is fine-tuned to output $y = 1$, indicating this collection does contain a hitting set of size ≤ 2 , where the hitting set contains diseased crops only.	151
6.15	Plant Hitting Sets task accuracy.	152
6.16	Plant Hitting Sets learned hypothesis.	152
6.17	Embed2Sym embedding spaces — Plant Hitting Sets.	153

6.18	CLEVR-Hans3 example data points. The meta-data M is empty. Class 1 images contain a large cube and a large cylinder. Class 2 images contain a small metal cube, and a small sphere. Class 3 images contain a large blue sphere, and a small yellow sphere.	154
6.19	CLEVR-Hans learned hypothesis.	154
6.20	Example mistake correction for the GPT-4 implementation of the Example Generator in the Two-Digit Arithmetic task. In this example, the mistake is using a <code>solution</code> predicate in the inclusion set, instead of the <code>label</code> predicate.	161

List of Tables

2.1	Truth table for the propositional connectives, where $\top = 1$ and $\perp = 0$	26
4.1	NSIL hyper-parameters used in the Cumulative Arithmetic tasks.	89
4.2	Cumulative Arithmetic task results.	90
4.3	Learning rate and momentum hyper-parameters used in the Two-Digit Arithmetic tasks.	92
4.4	Increasing the search space for the symbolic learner in the E9P task. * denotes the configuration used in the main results. Run-times in italics indicate that weight pruning is used to remove corrective examples with weight < 5 , to obtain results efficiently.	96
4.5	Two-Digit Arithmetic FF-NSL and NeurASP comparison. Best results highlighted in bold (we declare a tie in the case of overlapping error bars).	97
4.6	Learning rate and momentum hyper-parameters used in the Hitting Set tasks. . .	100
4.7	Increasing the search space for the symbolic learner in the Hitting Set FashionM-NIST task. * denotes the configuration used in the main results.	102
4.8	Hitting Set ablation results. Best results highlighted in bold (we declare a tie in the case of overlapping error bars).	102
5.1	End-to-end task accuracy on the Follow-Suit Winner task with decreasing architectural restrictions during training. With fewer architectural restrictions, accuracy increases.	126
6.1	CLEVR-Hans results.	154
6.2	Task accuracy with manual vs LLM engineering. The number of labels is decomposed as “num image labels + num task labels”. * denotes manual fixes to simple errors.	160
6.3	Foundation model zero-shot image accuracy.	162
6.4	Foundation model task accuracy with zero-shot image predictions.	162

Acronyms

AI Artificial Intelligence

ASP Answer Set Programming

CBM Concept Bottleneck Model

CNN Convolutional Neural Network

D2K Data-to-Knowledge

DST Dempster-Shafer Theory of Evidence

FCN Fully Connected Network

FF-NSL Feed-Forward Neuro-Symbolic Learner

FNR False Negative Rate

GPU Graphics Processing Unit

ILP Inductive Logic Programming

LAS Learning from Answer Sets

LLM Large Language Model

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MLP Multi-Layer Perceptron

NSIL Neuro-Symbolic Inductive Learner

RF Random Forest

SASR Semantic Aware Scene Recognition

VAE Variational Autoencoder

VQA Visual Question Answering

WCDPI Weighted Context Dependant Partial Interpretation

Notation

The following list only includes the notation that is recurrently used throughout the thesis.

Data Structures

a	a scalar (integer or real)
$\mathbf{a}$	a vector
$\mathbf{A}$	a matrix
$\mathcal{X}$	a set of elements
$X = \langle x_1, \dots, x_n \rangle$	an n -length sequence of elements
$ \mathcal{X} $	the number of elements of the set $\mathcal{X}$
$ X $	the length of the sequence X
$[a, b]$	the set of all real numbers between a and b inclusive, i.e., $\{x \in \mathbb{R} : a \leq x \leq b\}$
$\{a .. b\}$	the set of all integers between a and b inclusive, i.e., $\{x \in \mathbb{Z} : a \leq x \leq b\}$

Functions

$f : \mathcal{A} \rightarrow \mathcal{B}$	The function f with domain $\mathcal{A}$ and range $\mathcal{B}$
$\arg \min_x f(x)$	value of x at which $f(x)$ is minimised
$\arg \max_x f(x)$	value of x at which $f(x)$ is maximised

Learning from Answer Sets

B	the set of background knowledge of a learning task
H	the learned hypothesis (a set of rules)
$\mathcal{H}$	the hypothesis search space
E	a set of symbolic training examples
e	a symbolic training example, where $e = \langle e_{\text{id}}, e_{\text{pen}}, e_{\text{pi}}, e_{\text{ctx}} \rangle$
e_{id}	a unique identifier of e
e_{pen}	the weight penalty of e
e_{pi}	a partial interpretation which contains sets of inclusion (e_{inc}) and exclusion (e_{exc}) atoms of e
e_{ctx}	the set of contextual facts of e

Neuro-Symbolic Learning

$\mathcal{X}$	the set of all images
---------------	-----------------------

$\mathcal{Z}$	the set of all primitive concepts
$\mathcal{Y}$	the set of all downstream task labels
$\mathcal{M}$	the set of all meta-data
$\mathbf{x}$	an image in the set $\mathcal{X}$
z	a primitive concept in the set $\mathcal{Z}$
y	a downstream task label in the set $\mathcal{Y}$
m	an element of meta-data in the set $\mathcal{M}$
X	a sequence of images from $\mathcal{X}$
M	a sequence of meta-data from $\mathcal{M}$
Z	a sequence of primitive concepts from $\mathcal{Z}$
k	the number of primitive concept values
$\mathcal{D}_{\text{image}}$	a training set of images labelled with primitive concepts
$\mathcal{D}_{\text{task}}$	a training set of sequences of images and associated meta-data, labelled with a downstream task label
$\langle \mathbf{x}, z \rangle$	a sample from $\mathcal{D}_{\text{image}}$
$\langle X, M, y \rangle$	a sample from $\mathcal{D}_{\text{task}}$

Chapter 1

Introduction

Simulating human-level intelligence in machines has long been the goal of AI. Humans are able to seamlessly combine two cognitive abilities, learning from experience, and reasoning over what has been learned (Valiant, 2003). In today’s world, a large amount of effort has been dedicated to constructing systems that learn from experience, based upon artificial neural networks that can be trained using a set of example inputs and outputs. Whilst there was a substantial effort in the 80s and 90s on developing reasoning systems based upon explicit symbols, referred to as *symbolic AI*, or *good old fashioned AI* (Haugeland, 1989), these techniques have largely been neglected in recent times. For example, government, industry, and academia are frantically seeking to exploit the benefits of Large Language Models (LLMs), which are a particular flavour of neural networks capable of solving a wide range of tasks such as language translation, text summarisation, general question-answering, and code generation (OpenAI et al., 2023). However, many leading figures within computer science argue these architectures are fundamentally incapable of reasoning, since they are only trained to predict the next element in a sequence (e.g., the next word in a sentence). As well as the lack of explicit reasoning, the knowledge learned by these approaches is difficult to interpret, and it is impossible to guarantee certain behaviours, or constraint satisfaction, when such systems are deployed. This raises serious questions about the robustness of the technology, especially when considering deployments in more safety-critical applications.

Given this backdrop, a sub-field within AI called *Neuro-Symbolic AI* is continuing to gain popularity amongst researchers, as well as policy makers (Hitzler and Sarker, 2022; Garcez and Lamb, 2023). Neuro-Symbolic AI is seen as a mechanism to ensure robustness, fairness, accountability, and transparency within AI systems, all of which are key pillars within government policy and legislation, e.g., the European AI act (COM, 2021). Technically, Neuro-Symbolic AI aims to combine two paradigms together: Statistical machine learning, to learn from raw data (experience), and symbolic AI based upon computational logic, to reason over what has been learned. Not only does the field take inspiration from Leslie Valiant in his remarks about the direction for future intelligent systems (Valiant, 2003), but also from psychologist Daniel Kahneman, through his work on *Thinking, Fast and Slow* (Kahneman, 2011). Kahneman proposes a cognitive theory of the human brain based upon two systems, called *System 1*, and *System 2*. System 1 is concerned with rapid, impulsive and unconscious actions, such as solving $2 + 2 = ?$, turning your head when someone calls your name, perceiving common objects such as road signs, and understanding simple sentences. System 2 on the other hand, is concerned with slower, more deliberate thinking, often involving logical reasoning. For example, solving $987 \times 123 = ?$, counting the number of letters in a given sentence, and driving a car for the first time on the opposite side of the road. In his book, Kahneman positions his work in the context

of critical decision making. In these situations, if we as humans are not focused, our system 1 can sometimes take-over, and suggest spurious actions, when ideally, we should engage our system 2. Neuro-Symbolic AI can loosely be thought of as combining system 1 and system 2 style thinking. For example, once trained, a neural network can provide fast perceptions from the environment (system 1), and a logical reasoning step can recommend higher-level decisions, or actions, based on these perceptions (system 2). In contrast, it is under active debate whether the current popular forms of AI, such as LLMs, are capable of performing system 1 and system 2 thinking. Some argue LLMs are restricted to system 1 (Karpathy, 2023), whereas others claim with appropriate prompt engineering, LLMs can achieve system 2 (Kojima et al., 2022). In Neuro-Symbolic AI, the central open challenge is exactly *how* should one integrate system 1 and system 2, that is neural and symbolic computation? This is the primary focus within the field today.

1.1 Motivations

Many existing neuro-symbolic architectures rely upon a complete, and manually engineered knowledge base in order to perform logical reasoning. Given this knowledge, the focus is on training a neural network to extract primitive concepts from raw input data, with an emphasis on data efficiency, and utilising known background knowledge. This class of architectures is known as *neuro-symbolic reasoning*, since one has to reason over the knowledge to abduce an appropriate signal for training the neural network. Examples along this direction include DeepProbLog (Manhaeve et al., 2018), NeurASP (Yang et al., 2020), and Logic Tensor Networks (Badreddine et al., 2022). The central challenge with these approaches is the requirement for a complete knowledge base. In practice, this knowledge may be laborious to engineer, or may not be known.

Meanwhile, recent advancements in symbolic machine learning open the door for learning the symbolic knowledge, given an appropriate search space, and a set of training examples. In particular, the Learning from Answer Sets (LAS) systems have achieved impressive results when learning from *noisy* examples (Law et al., 2018). In the context of a neuro-symbolic integration, this capability is of significant interest, since neural networks may not extract primitive concepts with perfect accuracy. The LAS systems are capable of learning expressive knowledge in the form of Answer Set Programming (ASP), which is a declarative knowledge representation framework known for its expressivity, and ability to efficiently represent complex knowledge. The LAS systems can perform both observational, and non-observational predicate learning, where in non-observational predicate learning, the required predicates are not directly observable in the examples, and must be inferred through an invented predicate. Also, the FastLAS system (Law et al., 2020a) has achieved state-of-the-art results in terms of its scalability, and is able to learn knowledge from large search spaces. In contrast, other symbolic learning approaches struggle on some or all, of these fronts, i.e., they are unable to handle noisy inputs, they are limited in the expressivity of knowledge that can be learned, or they require a very restricted search space. All of these limitations have prevented the exploration of a robust neuro-symbolic architecture that can *learn* symbolic knowledge from raw data. We refer to such an architecture as one that performs *neuro-symbolic learning*, in contrast to *neuro-symbolic reasoning*, where the knowledge is given. As a result, in this thesis, we investigate the following research questions:

Research questions

1. Can exact symbolic knowledge be learned from raw data, using a symbolic learner?
2. Can symbolic knowledge be learned in a robust manner, to preserve the performance given noisy, or out of distribution data?
3. Can a neural network be trained simultaneously as abstract knowledge is learned from the network predictions, and can the learning of abstract knowledge help to train the neural network in a more data efficient manner?
4. Can we scale a neuro-symbolic learning framework to complex problems?

Compared to other neuro-symbolic approaches which learn knowledge in a differentiable manner, we choose to learn the knowledge symbolically, since this is the only way to preserve the interpretability, and optimality guarantees that the symbolic learner provides. Also, we focus on tasks that require perception using a neural network (where primitive concepts are extracted from imagery), and reasoning over the learned knowledge, using a symbolic component. We decide to focus on these tasks because they are an ideal test-bed for showcasing integrated system 1 and system 2 behaviour. It enables us to demonstrate the benefit of using an integrated neuro-symbolic approach, as opposed to a fully neural system. Furthermore, this direction enables us to position our work relative to others in the field, since this is a common focus area within the neuro-symbolic literature. Throughout this thesis, the learned knowledge maps primitive concepts extracted from the neural network to higher-level decisions. We often refer to the reasoning task as the *downstream task*, since during inference, reasoning occurs after perception in a pipeline fashion. Let us now outline the technical challenges encountered in our proposed research direction.

1.2 Technical Challenges

To integrate a symbolic learner with a neural network, there are numerous technical challenges to address:

1. How does one tolerate incorrect neural network predictions when learning symbolic knowledge? Are the LAS symbolic learners robust enough to tolerate significant levels of noise originating from a neural network, especially when the input data is outside the distribution of data used to train the neural network?
2. How can one integrate a symbolic learner within an end-to-end architecture, since the symbolic learner is not differentiable? Is it possible to learn complex symbolic knowledge and train a neural network simultaneously, without providing labels for the primitive concepts?
3. How can one scale a neuro-symbolic learning architecture, to the limits of what both components can support? When considering a joint perception and reasoning task, there is a combinatorial explosion in the possible values of the set of primitive concepts that the neural network can extract. This is known as the *symbol grounding problem*, which is the challenge of connecting real-world objects or concepts to a set of symbols (Harnad, 1990).

1.3 Contributions

To address these challenges, we first identify new problems and datasets with the desired level of complexity, since the lack of appropriate benchmarks is an open problem in the neuro-symbolic community. We introduce various options that showcase the benefits of an integrated neuro-symbolic learning architecture. For example, the Follow Suit Winner tasks studied in Chapters 3, and 6 require perception from playing card imagery, and reasoning over the rules of the Follow Suit card game. Also, the Hitting Sets tasks, studied in Chapters 4 and 6, require perception from images of hand-written digits (Chapter 4), and plant disease imagery (Chapter 6), whilst solving an NP-complete reasoning problem. This enables us to showcase the expressivity the LAS symbolic learners provide, compared to other forms of symbolic machine learning, whilst solving complex perception and reasoning tasks within a neuro-symbolic architecture. Given these materials, we develop four flavours of novel neuro-symbolic learning systems, which aim to address the aforementioned challenges. These contributions are:

Feed-Forward Neuro-Symbolic Learning Since the LAS symbolic learning systems have achieved state-of-the-art performance when learning from noisy examples (Law et al., 2018), we first explore a neuro-symbolic integration that can leverage *pre-trained* neural networks to extract primitive concepts from raw data, before the symbolic knowledge is learned using these concept predictions. This tackles challenge 1, because with this flavour of architecture, pre-trained neural networks may predict incorrectly, and potentially with high confidence, when the input data is subject to *distributional shifts*. These shifts occur when the input data originates from a different distribution than the distribution of data used to pre-train the network. The question is, can the LAS symbolic learners tolerate significant levels of noise originating from the neural network? To investigate this, we develop a novel method of transforming neural network predictions and confidence scores, into weighted training examples for the symbolic learner. We aggregate confidence scores that the neural network outputs for multiple images, into a unified weight, to bias the symbolic learner towards covering examples that contain correct concept predictions. We demonstrate that this method is able to learn accurate knowledge, even when large percentages of training data points are subject to distributional shifts. Our approach outperforms other hybrid integrations, where a neural network and a random forest are used as the symbolic learning component.

Iterative Neuro-Symbolic Learning To address challenge 2, our second contribution lifts the requirement for the neural network to be pre-trained, and trains both neural and symbolic components in an end-to-end fashion, without using any labels for the primitive concepts. The approach leverages recent developments in neuro-symbolic reasoning, where a neural network can be trained to extract primitive concepts, using a complete knowledge base, and a downstream label. We develop an iterative algorithm, that learns a candidate set of knowledge, which is used to train a candidate neural network. The novelty is the method of biasing the symbolic learner, to either refine or retain the learned knowledge throughout the iterations. To achieve this, we use a weighted set of corrective examples that balance exploration vs. exploitation of the hypothesis search space. The examples are weighted using the overall system performance at a given iteration, along with the neural network confidence score. We demonstrate that this approach is able to learn the correct knowledge, and train the neural networks to accurately extract primitive concepts, given a variety of perception and reasoning tasks. Also, we demonstrate this approach outperforms the closest neuro-symbolic competitor (Dai and Muggleton, 2021), as well as a range

of fully neural baselines.

Exploring a Decoupled Neuro-Symbolic Learning approach Despite achieving strong results, our end-to-end approach is challenging to scale to complex tasks, largely due to challenge 3, as there is a combinatorial explosion in the choice of values for primitive concepts. We therefore attempt to develop a more scalable architecture, by decoupling the training of the perception component from the reasoning task, using self-supervised learning. This involves clustering the raw input data, and developing methods to learn the symbolic knowledge given cluster predictions. Whilst we introduce novel methods along this direction, we ultimately face challenges in labelling the clusters with appropriate concept values, as well as scaling self-supervised learning to complex perception tasks, including object detection. Therefore, our final contribution proposes an alternative approach.

Scalable Neuro-Symbolic Learning We leverage recent developments within large vision-language foundation models to extract primitive concepts from raw data using Visual Question Answering (VQA). This approach revisits the idea of sequential training from our first contribution, and adopts a fine-tuning approach for neural network training with limited labelled data points. A sequential approach places minimal restriction on the symbolic learner, which helps to realise its full scalability, in terms of learning the knowledge for complex reasoning tasks. We propose a novel method which transforms natural language answers given by the foundation model, into training examples for the symbolic learner. We demonstrate that this approach achieves a significant performance gain in complex tasks, outperforming our other contributions, as well as many other neural and neuro-symbolic baselines. Furthermore, this approach requires very few labelled data points, and can leverage a LLM to reduce the engineering required to “glue” the neural and symbolic components together. This addresses challenge 3.

Our contributions significantly extend the application domain of symbolic machine learning using the LAS systems, enabling general, and interpretable knowledge to be learned from raw input data. Previously, only symbolic inputs were supported. Each of our proposed systems achieves state-of-the-art results in their respective domains, and we demonstrate our final contribution can outperform many different flavours of neuro-symbolic architectures when solving complex tasks. Let us now outline the empirical contributions in this thesis.

Summary of Empirical Contributions Throughout this thesis, we perform a variety of experiments to evaluate the contributions proposed. In Chapter 3, recall that a pre-trained neural network is used to extract primitive concepts from raw image data, and a LAS symbolic learner learns complex knowledge, given a set of predictions from the neural network. The question we seek to answer is whether a LAS symbolic learner can learn accurate symbolic knowledge, given input imagery from a distribution different from that used to pre-train the neural network. In a deployed environment, it is likely that the distribution of images used to pre-train the neural network won’t exactly match the distribution observed at the time of symbolic learning. We therefore design evaluation tasks that enable us to apply reasonable distributional shifts to the input imagery. The value of this experiment is demonstrating the robustness of LAS symbolic learning systems, when they are tasked with learning knowledge from real neural network predictions, in the presence of distributional shifts within the input images. The results indicate the LAS symbolic learners are able to learn accurate knowledge, even when large percentages of training examples are subject to distributional shifts. This experiment

demonstrates that our approach, which leverages the LAS symbolic learners, is suitable for real-world deployment, learning abstract knowledge from unstructured image data.

In Chapter 4, we propose a method for simultaneously training both the neural network and the symbolic learner in an end-to-end fashion, to eliminate the requirement for labelling the unstructured image data with primitive concepts. Our approach is the first system that successfully learns complex knowledge using the LAS systems ILASP (Law et al., 2020b) and FastLAS (Law et al., 2020a) whilst training a neural network to extract primitive concepts from raw data. We target our evaluation on the complexity and expressivity of the symbolic knowledge that can be learned, and investigate whether our approach can learn more expressive knowledge than that learned by other end-to-end systems. We also investigate the performance of our approach in the presence of large hypothesis spaces, to demonstrate its scalability. The results indicate our approach is able to train both neural and symbolic components to a performant level, whilst learning complex and expressive knowledge that other end-to-end systems cannot learn.

The end-to-end learning of complex knowledge from unstructured data becomes challenging when the number of primitive concepts is large. End-to-end systems often rely on back-propagating information through architectural layers (either neural or symbolic), and if the number of primitive concepts is large, there could be many combinations of inputs that lead to a particular output, and therefore the gradient used in back-propagation is effectively shared between many possibilities. We investigate this by merging the image complexity from the tasks in Chapter 3, with the symbolic complexity from the tasks in Chapter 4. In Chapter 6 we propose an approach that leverages a fine-tuned vision-language foundation model for extracting primitive concepts from raw data, with the aim of increasing the difficulty of the perception task, whilst retaining the complexity and expressivity of the tasks to be solved with the symbolic learner. Empirically, we investigate whether our system can train both components to a performant level, whilst learning with very few labelled data points. Our sequential training approach enables us to investigate whether our system can solve tasks with a large number of primitive concepts to be extracted by the neural network. We achieve state-of-the-art performance across a variety of metrics, including accuracy, data efficiency, and engineering effort. Finally, since we leverage a vision-language foundation model, we also evaluate our approach on a task that requires visual object detection, instead of just image classification, as is the case for all other experiments in this thesis.

In summary, all the evaluation tasks presented in this thesis require primitive concepts to be extracted from raw image data using a neural network, and the learning of symbolic knowledge to solve a reasoning task using a symbolic learner. The differences are apparent when considering the architecture of each proposed approach. In Chapter 3 the neural network is pre-trained in full with a complete labelled dataset, in Chapter 4 the neural network is trained simultaneously alongside the symbolic learner without primitive concept labels, and in Chapter 6 the neural network is a vision-language foundation model that is fine-tuned with a small number of primitive concept labels. In Chapter 3 the evaluation targets the robustness of the symbolic learner, assuming full access to label information for training the neural network. In Chapter 4 we assume no primitive concept labels are available, and the neural network has to be trained using candidate knowledge learned by the symbolic learner, as well as the final labels of the reasoning task. Finally, in Chapter 6 our evaluation assumes an overall minimal set of labels when considering the labels required to train both the neural and symbolic components. The evaluation targets a broad set of tasks, including tasks with more challenging perception requirements such as object detection, as well as tasks that require learning complex and expressive symbolic

knowledge.

1.4 Publications

The work in this thesis has appeared in the following publications:

- **Daniel Cunningham**, Mark Law, Alessandra Russo, Jorge Lobo, Lance Kaplan. Towards Neural-Symbolic Learning to Support Human-Agent Operations. *In proceedings of the 24th IEEE International Conference on Information Fusion (FUSION)*. Pages 1–8, 2021.
- **Daniel Cunningham**, Mark Law, Jorge Lobo, Alessandra Russo. Feed-Forward Neural-Symbolic Learner. *In Machine Learning Special Issue on Learning and Reasoning*. Volume 112, pages 515–569, 2023. This work was also presented at the 2022 International Joint Conference on Learning and Reasoning.
- **Daniel Cunningham**, Mark Law, Jorge Lobo, Alessandra Russo. Neuro-Symbolic Learning of Answer Set Programs from Raw Data. *In proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI-23)*. Pages 3586–3596, 2023. This work was also selected for a spotlight talk at the 2022 AAAI Fall Symposium titled “*Thinking Fast and Slow and Other Cognitive Theories in AI*”, and presented at the 2023 International Joint Conference on Learning and Reasoning.
- **Daniel Cunningham**, Mark Law, Jorge Lobo, Alessandra Russo. The Role of Foundation Models in Neuro-Symbolic Learning and Reasoning. *In proceedings of the 18th International Conference on Neural-Symbolic Learning and Reasoning (NeSy 2024)*. Pages 84–100, 2024. This work was selected for a spotlight talk at NeSy 2024.

Collaborations outside of this thesis have resulted in the following workshop papers:

- **Daniel Cunningham**, Flaviu Cipcigan, Rodrigo Neumann Barros Ferreira, Jonathan Booth. Symbolic Learning for Material Discovery. *Presented at the AI for Accelerated Materials Discovery workshop, Neural Information Processing Systems Conference (NeurIPS)*, 2023.
- Jack Furby, **Daniel Cunningham**, Dave Braines, Alun Preece. Towards a Deeper Understanding of Concept Bottleneck Models Through End-to-End Explanation. *Presented at the Representation Learning for Responsible Human-Centric AI (R2HCAI) workshop, Association for the Advancement of Artificial Intelligence Conference*, 2023.

1.5 Thesis Structure

This thesis is organised as follows. In Chapter 2, we present an overview of the technical background materials required to understand the contributions of this thesis. We discuss the relevant techniques from the fields of deep learning, and symbolic AI. The contributions of this thesis are presented in four chapters, as follows:

- In Chapter 3, we present our first contribution towards an integrated neuro-symbolic learning system, called Feed-Forward Neuro-Symbolic Learner (FF-NSL), which leverages *pre-trained* neural networks.

- In Chapter 4, we present our second contribution, called Neuro-Symbolic Inductive Learner (NSIL), which relaxes the requirement for the neural network to be pre-trained, as both neural and symbolic components are trained simultaneously.
- In Chapter 5, we explore a decoupled approach to neuro-symbolic learning, using techniques from the field of self-supervised learning. The idea is to improve scalability by training the network independently, and also without labelled concept information.
- In Chapter 6, we present our final contribution of scaling neuro-symbolic learning with foundation models.

We conclude with a discussion of the related work in Chapter 7, followed by a summary of the contributions and findings in Chapter 8, where we also outline directions for future work. The appendices contain verbose code listings regarding the background knowledge and learning task specifications used in the various tasks throughout this thesis.

Chapter 2

Background

In this chapter, we outline the key technical building blocks of this thesis. Since neuro-symbolic AI is concerned with the intersection of both neural and symbolic computation, we present the relevant background material from each discipline. We first present material from the field of deep learning, which is concerned with purely neural computation. Second, we provide an overview of the relevant material from the field of symbolic AI, including computational logic, Inductive Logic Programming (ILP), ASP, and the LAS symbolic learning framework this thesis builds upon.

2.1 Deep Learning

With the advent of accelerated Graphics Processing Unit (GPU) hardware, the field of deep learning has grown significantly over the last decade, and underpins many of the recent publicised achievements of AI. Deep learning is an extension of artificial neural networks, which aim to simulate the neuron pathways in the human brain. In the early days, research was focused on simulating individual neurons (McCulloch and Pitts, 1943), followed by architectures that combine neurons together into layers (Rosenblatt, 1958). Today, the central idea behind *deep* learning, is to combine many layers of neurons together within an artificial neural network, to enable complex and non-linear representations to be learned. In this section, we present the fundamental deep learning concepts, inspired by the material presented in (Russell and Norvig, 2002; Goodfellow et al., 2016; Kruse et al., 2022; Cingillioglu, 2022).

2.1.1 Artificial Neural Networks

Artificial neural networks, referred to hereafter as simply neural networks, are universal function approximators that map real-valued input vectors to real-valued output vectors. The input and output vectors can represent anything that is possible to encode in a real-valued form. For example, continuous input data such as the acidity, pH level, and the amount of alcohol in a bottle of wine can be mapped to its quality (Cortez et al., 2009). Also, pixels representing an image can be encoded into a vector, which is mapped to an output vector that denotes probability assignments over certain classes. Categorical data can also be encoded into a real-valued form using the one-hot encoding technique, where assignment over k values is denoted by a vector of length k , with an entry of the value 1 in the index that represents the particular category value, and 0 otherwise. Natural language can also be represented using one-hot encoding, where the vector length is equal to the number of words in a given vocabulary, and the index in the

vector with the value 1, represents the word at this index in the vocabulary. To avoid numerical overflows, a popular approach is to normalise the real-valued input into the $[0, 1]$ range, using for example, the min-max technique. Given an input vector $\mathbf{x} \in \mathbb{R}^n$, a normalised vector $\mathbf{x}'$ can be computed as:

$$\mathbf{x}' = \frac{\mathbf{x} - \mathbf{x}_{\min}}{\mathbf{x}_{\max} - \mathbf{x}_{\min}} \quad (2.1)$$

where $\mathbf{x}_{\min}$ and $\mathbf{x}_{\max}$ refer to the minimum and maximum values in the vector respectively.

The goal of a neural network is to approximate an unknown function f^* , by learning a mapping $y = f(\mathbf{x}; \boldsymbol{\theta})$ that outputs a desired target y , given the input vector $\mathbf{x}$. In the case of a classification task, y can denote the category of the input $\mathbf{x}$, or in the case of a regression task, y can represent a real-valued number. The neural network is parametrised by $\boldsymbol{\theta}$, and the objective is to learn these parameters such that the correct outputs are obtained for every input in a given training dataset. A popular neural network architecture is the Multi-Layer Perceptron (MLP), which flows information through the network in a feed-forward fashion, i.e., the network can be thought of as a directed acyclic graph, with no loops.¹ In a feed-forward architecture, the learned mapping function f is a composition of functions $y = f(\mathbf{x}; \boldsymbol{\theta}) = f^L \circ f^{L-1} \circ \dots \circ f^1(\mathbf{x}; \boldsymbol{\theta}_1)$, which represents the compositional structure across different layers. The total number of layers is denoted by L , and the final output can be computed by chaining the functions at each layer together. The first layer processes the input vector $\mathbf{x}$, and subsequent layers process the output of the previous layer. f^1 is referred to as the input layer, f^l , where $1 < l < L$ are referred to as the hidden layers, and f^L is referred to as the output layer. The number of layers in the architecture corresponds to the depth, hence the term *deep* learning is used to describe neural networks with many hidden layers. A MLP consists of *linear* layers, where each layer consists of multiple neurons which act in parallel. Also, artificial neural networks connect each neuron in layer $L - 1$, to each neuron in layer L , which gives rise to the term *fully-connected* network. The idea is to simulate neurons in the human brain, which receive inputs from many other neurons and compute their own activation value. Assuming a layer f^l contains d_l neurons, f^l takes the form:

$$f^l(\mathbf{z}^{l-1}; \boldsymbol{\theta}^l) = \mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l = \mathbf{z}^l \quad (2.2)$$

where $\mathbf{z}^{l-1} \in \mathbb{R}^{d_{l-1}}$ is the output of the previous layer with d_{l-1} neurons, $\mathbf{W}^l \in \mathbb{R}^{d_l \times d_{l-1}}$ is a learnable weight matrix for the current layer, and $\mathbf{b}^l \in \mathbb{R}^{d_l}$ is a learnable bias vector, which enables the network to learn functions that are shifted from the origin. Note the learnable parameters of each layer are denoted by $\boldsymbol{\theta}^l = \{\mathbf{W}^l, \mathbf{b}^l\}$. A basic linear layer can only learn linearly separable functions, which restricts the complexity of knowledge that can be learned. Even if multiple layers are stacked together, the overall computation reduces to a linear calculation and is therefore equivalent to a single layer. To enable the network to learn more complex functions, a *non-linear* activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is required:

$$f^l(\mathbf{z}^{l-1}; \boldsymbol{\theta}^l) = \sigma(\mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l) = \mathbf{z}^l \quad (2.3)$$

In this case, stacking multiple layers together with non-linear activation functions enables highly complex functions to be learned, and the full learning potential of the neural network can be realised. Although the activation can be computed by any non-linear differentiable function, there are common functions used in the literature:

¹Neural networks that have loops are called *recurrent neural networks*.

- **Sigmoid.** The sigmoid function, also known as the logistic function, is an S-curve that produces an output in the range $[0, 1]$. It is particularly useful in the output layer for binary classification tasks, since the output can be interpreted as a probability, and compared to a threshold value to decide whether the prediction is 0, or 1.

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \quad (2.4)$$

- **Softmax.** The softmax function is a generalisation of the sigmoid function to multiple outputs, often used in the output layer for multi-class classification tasks. The softmax function outputs a probability distribution that can represent class assignment probabilities. For example, assuming k classes, the activation (or probability) for class i can be computed as:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{j=1}^k e^{\mathbf{x}_j}} \quad (2.5)$$

- **tanh.** The hyperbolic tangent function is a scaled version of the sigmoid function that has output in the range $[-1, 1]$. This ensures negative inputs are mapped to negative outputs. Tanh can be used as the activation function of a hidden layer, and since it is centred on zero, it can help the network learn with normalised input data. Tanh is defined as:

$$\tanh(\mathbf{x}) = 2\sigma(2\mathbf{x}) - 1 \quad (2.6)$$

- **ReLU.** The Rectified Linear Unit activation function is one of the most popular activation functions to use in the hidden layer, because it is computationally efficient, and is less susceptible to the *vanishing gradient* problem (continue reading for an explanation). The ReLU activation function is defined as:

$$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x}) \quad (2.7)$$

So far, we have defined the architecture of the MLP, which consists of multiple linear layers composed together, using non-linear activation functions. The neural network parameters θ are initialised randomly, and therefore, the initial outputs are likely to be incorrect. As a result, the network parameters need to be learned, which is often achieved using an algorithm called *gradient descent*. Gradient descent updates the parameters based upon an error signal calculated by comparing the network prediction $\hat{y}$ with a ground-truth target label y . The error signal is generally computed using a loss function, which is computed using a dataset $\mathcal{D} = \{\langle \mathbf{x}_1, y_1 \rangle, \dots, \langle \mathbf{x}_n, y_n \rangle\}$ of labelled training data points. The optimisation objective is to minimise the loss function $\mathcal{L}$ across the training set:

$$\theta^* = \arg \min_{\theta} \mathcal{L}(\mathcal{D}; \theta) \quad (2.8)$$

The specific loss function used is highly task dependant, and must be hand-selected during the engineering of the training architecture. Common options include:

- **Mean Absolute Error (MAE).** This is often used in regression tasks, to measure the absolute difference between the predicted and target values.

$$\mathcal{L}(\mathcal{D}; \theta) = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i| \quad (2.9)$$

- **Mean Squared Error.** Also used in regression tasks, this penalises the network for making large errors, by squaring them. The drawback is vulnerability to outliers, or noise, as incorrect data points result in a large (incorrect) effect on the network parameter updates.

$$\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (2.10)$$

- **Binary Cross-Entropy.** This is commonly used in binary classification tasks.

$$\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)) \quad (2.11)$$

- **Categorical Cross-Entropy.** This extends the Binary Cross-Entropy loss for multi-class classification tasks. It is also known as the negative log-likelihood loss. In Chapter 3, when neural networks are trained using direct supervision, this is the loss function used. Assuming k classes, the Categorical Cross-Entropy loss is defined as:

$$\mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = - \sum_{i=1}^k y_i \log(\hat{y}_i) \quad (2.12)$$

Crucially, the loss function has to be differentiable, in order for gradient descent to be used. The gradient of the loss function with respect to all the trainable parameters is computed, which is calculated as the partial derivatives of each parameter:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) = \left(\frac{\partial \mathcal{L}(\mathcal{D}; \boldsymbol{\theta})}{\partial \theta^1}, \dots, \frac{\partial \mathcal{L}(\mathcal{D}; \boldsymbol{\theta})}{\partial \theta^n} \right) \quad (2.13)$$

The partial derivative of each parameter can be calculated using the chain rule, since all of the operations within the network are differentiable:

$$\frac{\partial \mathcal{L}(\mathcal{D}; \boldsymbol{\theta})}{\partial \theta^i} = \frac{\partial \mathcal{L}(\mathcal{D}; \boldsymbol{\theta})}{\partial \mathbf{z}^{i+1}} \frac{\partial \mathbf{z}^{i+1}}{\partial \mathbf{z}^i} \frac{\partial \mathbf{z}^{i+1}}{\partial \theta^i} \quad (2.14)$$

where $\boldsymbol{\theta}^i$ and $\mathbf{z}^i$ are the parameters, and output of the i -th layer, respectively. The gradient of the loss function with respect to the output of the $i + 1$ th layer is calculated, before back-propagating through the preceding layers to compute the gradient with respect to the i th layer. In *deep* neural networks, where many layers are stacked together, there can be many derivatives to calculate. This leads to the aforementioned vanishing gradient problem, which occurs when many derivatives in a deep network are multiplied together, leading to an increasingly small gradient. The ReLU activation function mitigates this, since it outputs 0 for negative inputs. In this case, the gradients are also 0, and therefore do not vanish. A gradient of 0 however, ultimately prevents any learning, so no weight updates are performed. This can be harmful, or beneficial, for similar reasons to the popular Dropout regularisation technique, which prevents over-fitting by disabling certain neurons (Baldi and Sadowski, 2013). To perform gradient *descent*, where parameter updates are applied to *minimise* the loss function, the following update rule is used:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) \quad (2.15)$$

where t is the iteration number, and α is a learning rate that controls the magnitude of the weight updates. The goal is to train the network over a number of iterations until the gradients become close to 0 and the parameters converge. A larger learning rate can accelerate training,

but can also overshoot the target. There are therefore many variations of the weight update presented in Equation 2.15, that for example, perform various annealing schedules, where the learning rate is initially set to a high value, and then reduced throughout training. Popular variants of the gradient descent algorithm also include the Adam optimiser (Kingma and Ba, 2014), which maintains a per-parameter learning rate that is adapted throughout training. In contrast, the Stochastic Gradient Descent (SGD) optimiser is another technique that maintains the same learning rate for all parameters. It has been shown that although Adam is faster than SGD, SGD can generalise better on certain problems (Wilson et al., 2017). In practice, the choice of optimiser, learning rate, annealing schedule, collectively form *hyper-parameters* which must be optimised and tuned as part of the engineering process. In this thesis, unless otherwise stated, we use the SGD optimiser from the PyTorch library² as empirically, this results in the best performance in our experiments.

One of the key challenges in optimising a neural network is whether the loss function is convex or not. A convex loss function has a single global minimum, which makes the loss function easy to optimise, since gradient descent is guaranteed to find the global minimum, as the gradient always points in the direction of the steepest descent. However, in many applications, the loss function used is non-convex, and thus, the network is vulnerable to becoming trapped in a local minima. This poses a challenge in practice, and the network architecture, the hyper-parameters, and various regularisation techniques are tuned and experimented with in a trial-and-error process, in order to maximise the network performance. This is usually evaluated on a separate dataset called a validation set, to avoid over-fitting the parameters to the training set. The network configuration may also influence whether it *over-fits* to the training set. A network is said to over-fit, if the performance on a held-out test set plateaus, or even decreases, whilst the performance on the training set continues to increase. Common remedies for this include obtaining a larger and more varied training set, reducing the complexity of the network architecture, or using regularisation techniques such as dropout (Baldi and Sadowski, 2013).

In contrast, the symbolic learning systems utilised in this thesis are able to learn *general* knowledge, that does not over-fit to the training data, provided enough examples are given.³ This is because the symbolic learners employ an optimisation heuristic based upon the length of the learned programs, to encourage shorter, and therefore more general programs to be learned. Finally, in terms of supervision, neural networks can be trained using a variety of approaches. The most popular approaches are:

- **Fully Supervised.** The network is provided with explicit labels for each piece of input data. In Chapter 3, we use this approach to pre-train our neural networks.
- **Transfer Learning.** It has been shown that earlier layers in a neural network learn more general features, whereas the later layers learn more specific knowledge related to the training dataset (Simonyan et al., 2013). It is possible to *transfer* some of this learned knowledge by taking a pre-trained neural network, freezing the weights of the early layers (this disables any weight updates, but still enables gradients to flow through), and “fine-tuning” only later layers for the task at hand. This often reduces the amount of data required compared to a fully supervised approach that trains the network from scratch. We adopt this approach in Chapter 6.
- **Weakly Supervised.** This is common in the neuro-symbolic literature, where a neural network is trained to perform a classification task, via a reasoning component that requires

²<https://pytorch.org/docs/stable/generated/torch.optim.SGD.html>

³The number of examples required is significantly less than what is required to train a deep neural network.

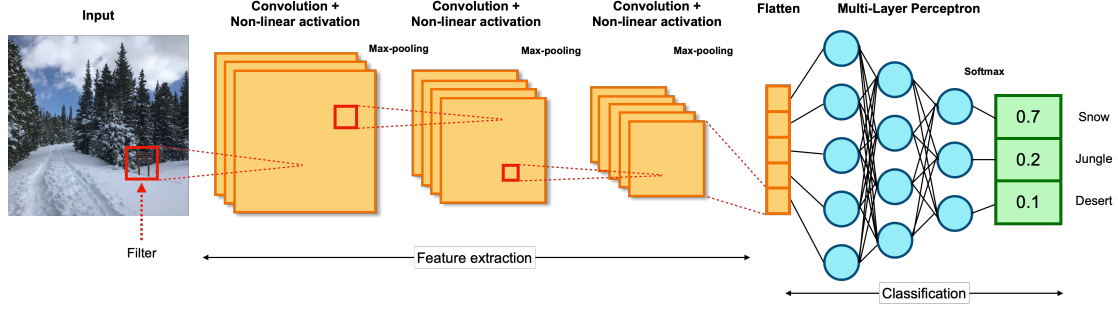


Figure 2.1: Example CNN architecture with three convolution layers, when tasked with classifying an image as “Snow”, “Jungle”, or “Desert”.

label information only for a downstream task. For example, a perception neural network may classify handwritten digits, despite label information being provided only for the sum of a pair of digits. In this case, the reasoning component abduces a suitable training signal for the perception network. We adopt this approach in Chapter 4, where we develop an end-to-end neuro-symbolic learning system.

- **Self-Supervised.** This is a recent approach that produces an appropriate supervision signal using the data alone. For example, in image classification, there are self-supervised approaches that encourage the network to predict the same output for various transformations of the same image, and different outputs for transformations of different images. In Chapter 5, we explore whether a self-supervised approach could be integrated within a neuro-symbolic learning architecture, to eliminate the requirement to label the raw inputs with primitive concept annotations.
- **Unsupervised.** In Unsupervised learning, there is no supervision at all, for example, in clustering, where the neural network learns an appropriate representation just by observing the data alone. Common techniques in this category are autoencoders (Kramer, 1991), and variational autoencoders (Kingma and Welling, 2013), which we explore in Chapter 5.

2.1.2 Advanced Neural Network Architectures

Neural network architectures are inherently flexible, since they enable different components to be connected together. It is possible to construct bespoke architectures for a particular class of problems, that leverage specific biases and features of the problem domain. As well as the MLP introduced so far, in this thesis, we use a variety of neural network architectures as baselines, and as part of our proposed methods. Let us now describe the architectures used.

Convolutional Neural Networks A popular architecture is the Convolutional Neural Network (CNN), which has made a significant impact in the machine learning community (Li et al., 2021). CNNs exploit the translational invariance of the input data, which is particularly useful when dealing with spatial data such as 1-D time series, and 2-D imagery, since relevant features may appear in any position within the input data. Although the CNN architecture was originally published in the 80s (Fukushima, 1980; LeCun et al., 1989), it was only in the 2010s that CNNs became widely used in deep learning, given the advent of GPU hardware accelerations. In fact, it was CNNs themselves that fuelled a resurgence of interest in neural networks in general, when a deep CNN won the prestigious ImageNet Large Scale Visual Recognition Challenge in 2012 (Krizhevsky et al., 2012).

The central operation performed, as the name suggests, is the convolution operation. In the neural network architecture, the convolution operation is performed within a convolutional layer, which consists of a set of learnable filters (also called kernels). Each filter is a matrix of weights, which is convolved over the input data, to extract relevant features. One can think of the filter as a sliding window, which is moved over the input data (in the case of 2-D images) from the top left to the top right. At each window position, a linear layer operation is performed, that multiplies the filter weights by the region in the input that overlaps with the sliding window. These multiplications are then summed to produce an entry in the convoluted output matrix. The filter is then moved to the right, and the process is repeated. Once the filter reaches the end of a “row” in the image, the filter moves to the left most position of the next “row”, where a row is determined by the filter size with respect to the input data (e.g., the pixels in the image). After each convolution operation, a non-linear activation is applied, such as ReLU, enabling the network to learn complex relationships and spatial patterns within the data. Once the convolved matrix is computed, a typical next step is to perform pooling, or downsampling. These pooling layers reduce the size of the convoluted matrix, usually by taking the maximum, or an average value, within a given region. This enables performance benefits during training, whilst still retaining key information. The pooling step can be thought of as a data compression technique. Multiple convolution and pooling layers are stacked on top of each other, to enable a range of features to be learned. The earlier layers may learn to detect edges, colours, and basic shapes, whereas later layers may learn to detect higher level features such as the wheel of a car, or the face of a dog. The next stage in the architecture flattens the output of the convolutional and pooling layers into a 1-D vector, which can then be passed to a MLP to be mapped to the desired output vector. An overview of the CNN architecture is shown in Figure 2.1.

In practice, there are many hyper-parameters that can be tuned to find the optimal CNN architecture for a given task. These include:

- **Filter size.** This defines the size of the sliding window in the convolutional layer. Smaller filters are generally preferred, so they can focus on learning representations over local features within the data. Also, odd-sized filters are generally preferred, since they have a well-defined centre pixel. This maintains symmetry, and preserves the spatial dimensions of the input data in the convolved matrix. If an even-sized filter is used, the feature maps can be mis-aligned with the input data. Therefore, a filter size of 3×3 , or 5×5 is generally chosen in practice.
- **Stride length.** This determines how many “cells” (pixels in the case of images) in the input data to skip when sliding the filter to the next region. This is used to reduce the amount of computation in cases where the input data does not vary too greatly between cells.
- **Number of filters.** Increasing the number of filters enables the network to learn additional feature representations. These filters are initialised with different weights.
- **Padding.** This is concerned with the edges of the input data where the filter may only partially cover the data. In this case, the filter can be padded with zeros, or edge regions can be ignored entirely.

The CNN architecture can then be trained end-to-end using the standard gradient descent algorithm introduced in Section 2.1.1. In this thesis, we use a CNN as our perception component in Chapters 3 and 4. We integrate a CNN with a symbolic learner, to enable primitive concepts

to be extracted from raw image data, before learning knowledge that maps the concepts to downstream predictions. One disadvantage of both the CNN, and MLP architectures, is that they struggle to deal with sequence data, and in particular, long sequence data, since there are no loops, or feedback connections. Without such connections, it is difficult for the network to use historical information from earlier in the sequence when considering, for example, the task of predicting the next element in the sequence. *Recurrent* neural network architectures have been proposed to tackle this problem (Rumelhart et al., 1986).

Recurrent Neural Networks A recurrent neural network accepts a sequence of data as input. We refer to each item in the sequence as being observed at a particular time step t . Note that each time step often may not correspond to the timestamp of a real-world observation, as t is also used to simply denote the index of the element in the sequence. The basic idea behind the recurrent neural network is to modify the computational graph to form loops, or, recurrent connections, where the output of a layer at a particular time step becomes one of the inputs to itself at the next time step. The linear layer from Equation 2.2, can be extended as follows, through the addition of a hidden state vector $\mathbf{h}$:

$$f^t(\mathbf{h}^{t-1}, \mathbf{x}^t; \boldsymbol{\theta}) = \mathbf{W}\mathbf{h}^{t-1} + \mathbf{U}\mathbf{x}^t + \mathbf{b}^t \quad (2.16)$$

where $\mathbf{W}$ and $\mathbf{U}$ are weight matrices applied to the hidden vector from the previous time step $\mathbf{h}^{t-1}$, and the input to the current time step $\mathbf{x}^t$, respectively. The hidden vector is what forms the loop, as the output of the network at state t , not only depends on the input at the current time step, but also the hidden vector from the previous time step. For example, if a recurrent neural network were to process words in the input sequence “I love this thesis”, when the network observes the word “love”, it receives an embedding vector representing the word “love” as input,⁴ and also the hidden vector from the previous time step when the word “I” was observed. This enables the network to capture dependencies between elements in the sequence, as information related to previous elements is available when processing the current element. Also, the recurrent connection enables infinite, and variable length sequences to be observed, whereas an MLP requires the dimensions of the input to be defined apriori. In a similar fashion to the CNN, multiple recurrent layers can be stacked on top of each other to form deeper networks, and learn more abstract representations. The entire architecture can be trained via back-propagation and gradient descent, as discussed in Section 2.1.1. However, recurrent neural networks are difficult to apply to long sequences, due to the vanishing gradient problem. Since the input at time step t requires the hidden vector from time step $t - 1$, which itself requires the input from $t - 2$, etc..., there are many gradients to back-propagate. Through repeated application of the chain rule in Equation 2.14, the successive multiplications leads to extremely small gradients, and eventually the network is unable to learn, since the gradients are very close to 0. Therefore, the Long Short-Term Memory (LSTM) network has been proposed to tackle this problem.

LSTM The core idea behind the LSTM is the *cell state*, which acts like an information conveyor belt. Through a series of linear interactions, the cell state enables the LSTM to learn whether to add information, remove information, or simply allow information to pass through to the next time step. These linear interactions are called *gates*, and are composed using a sigmoid function

⁴An embedding layer is often used to encode raw input data such as natural language into a vector representation. One-hot encoding can be used, as well as more advanced techniques such as Word2Vec (Mikolov et al., 2013a).

with a point-wise multiplication operation. The sigmoid function is used since the output is the range $[0, 1]$, so it becomes easy to interpret as a Boolean decision. Let us now explain the various gates. First, we start with the *forget gate* which decides which information from the previous time step should be omitted. The forget gate is defined as:

$$\mathbf{f}^t = \sigma(\mathbf{W}^f [\mathbf{h}^{t-1}, \mathbf{x}^t] + \mathbf{b}^f) \quad (2.17)$$

where $\mathbf{h}^{t-1}$ is the hidden vector from the previous time step, $\mathbf{x}^t$ is the input at the current time step, and $[\ , \]$ denotes vector concatenation. $\mathbf{W}^f$ and $\mathbf{b}^f$ represent the weights and biases respectively, and σ is the sigmoid function. The sigmoid function outputs a vector of length equal to the hidden vector from the previous state, and each entry in the vector denotes whether the information should be forgotten (0) or retained (1). The next step is to compute the input gate, which decides which elements of the input vector should be added to the current state. This requires two gates. First there is an input gate $\mathbf{i}^t$ that decides which values should be added, and second, there is the cell input gate $\tilde{\mathbf{c}}^t$ which computes the values themselves:

$$\mathbf{i}^t = \sigma(\mathbf{W}^i [\mathbf{h}^{t-1}, \mathbf{x}^t] + \mathbf{b}^i) \quad (2.18)$$

$$\tilde{\mathbf{c}}^t = \tanh(\mathbf{W}^c [\mathbf{h}^{t-1}, \mathbf{x}^t] + \mathbf{b}^c) \quad (2.19)$$

Note that the input gate takes the same form as the forget gate since this is performing a similar filtering operation. At this stage, the new cell state can be computed. This involves applying the forget gate to remove the required information, achieved through multiplication (recall a 0 value from the forget gate indicates this information should be omitted), and applying the input gate to include the required new information, achieved through addition. The new cell state is defined as follows:

$$\mathbf{c}^t = \mathbf{f}^t \times \mathbf{c}^{t-1} + \mathbf{i}^t \times \tilde{\mathbf{c}}^t \quad (2.20)$$

The output gate then decides which parts of the current state should be returned as output to the next cell state. This is achieved using two steps. First, a sigmoid filter is performed again to decide which elements should be output from the current state, and second, this filter is applied to the current state, to create the hidden state that is input to the next cell state. The output gate $\mathbf{o}^t$, and the next hidden state $\mathbf{h}^t$, are defined as follows:

$$\mathbf{o}^t = \sigma(\mathbf{W}^o [\mathbf{h}^{t-1}, \mathbf{x}^t] + \mathbf{b}^o) \quad (2.21)$$

$$\mathbf{h}^t = \mathbf{o}^t \times \tanh(\mathbf{c}^t) \quad (2.22)$$

where the $\tanh$ function normalises the current cell state into the range $[-1, 1]$. An overview of the gate connections and information flow in an LSTM cell is shown in Figure 2.2. In this thesis, we leverage the LSTM recurrent architecture as a baseline model in Chapters 3 and 4, when dealing with sequence data.

Despite the success of LSTM neural networks in a wide variety of applications, it can be computationally expensive to train with large datasets. This is because the LSTM, and the vanilla recurrent neural network upon which it is based, processes data sequentially. Sequential processing fails to take advantage of the parallelism offered by modern computing, and in particular GPUs. This bottleneck has led to the invention of the transformer architecture, which

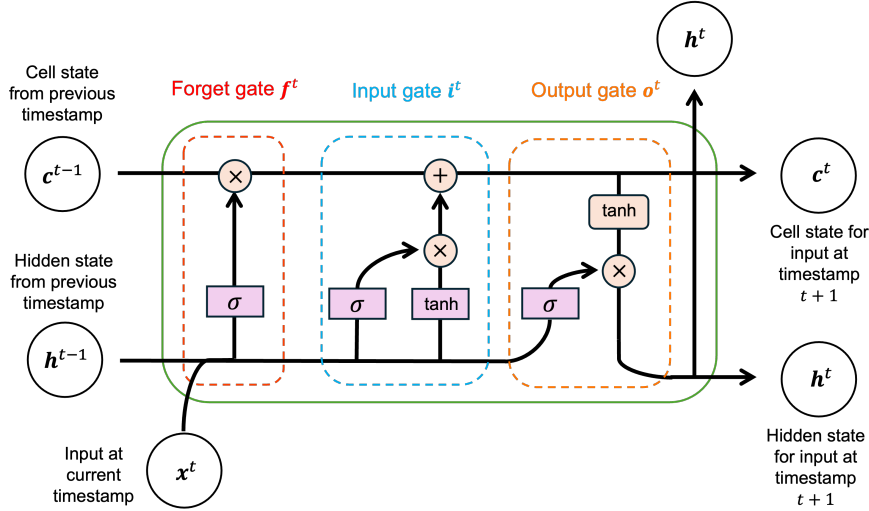


Figure 2.2: Visual overview of the information flow within an LSTM cell.

processes sequence data in a parallel fashion, using neural network components called *encoders* and *decoders*. The ability to refer to historical parts of the input sequence is handled by a mechanism called *attention*, which is designed to retain the recurrent benefits of the LSTM. Let us now describe the transformer architecture.

Transformer The transformer, first introduced in (Vaswani et al., 2017), has achieved significant success in scaling deep learning to complex problems, and since its introduction, has become the go-to architecture for a wide range of tasks. At a high-level, the architecture consists of a set of encoder and decoder layers, stacked together. The encoder maps the input sequence to a series of continuous representations, in a high-dimensional embedding space, and the decoder takes the output of the encoder, along with the output of the decoder at the previous time step, and generates the next output. The transformer was initially designed for machine translation, where the encoder generates a general semantic embedding for the source language, and the decoder outputs the same semantic representation in the target language. The encoder and decoder consist of modules stacked on top of each other, and include feed-forward MLP components, as well as the novel attention mechanisms that enable the architecture to incorporate historical information. Crucially, this architecture can be parallelised, which greatly reduces the computational cost of training. Let us now describe the architecture.

The first step is to encode the input data into a continuous representation, using an embedding layer. Since transformers do not have the concept of recurrence, a *positional encoding* is also generated, which is a vector that denotes the position of each piece of data in the input sequence. The bottom-most encoder in the stack then accepts the embedding vector, and the positional encoding as input, and employs *self-attention*, as well as feed-forward layers, to generate the output. The self-attention mechanism enables the encoder to focus on different parts of the input sequence, when it is processing each element, often referred to as a *token*. This is similar to the recurrence in the LSTM architecture, except in the transformer, each historical element in the sequence is directly available, rather than having to be unrolled from the immediate previous element. Essentially, the attention layer computes a score, which denotes the importance of each previous token when processing the current token. The attention scores are fed through a softmax layer to create an attention weight matrix, which is multiplied by the input values to yield an output with the key information. To mitigate the effect of the vanishing

gradient problem, normalisation and residual connections are used. To achieve this, the input to the attention layer is concatenated with its output, and then normalised. This provides a “short-cut” for gradients to flow during back-propagation, which ensures earlier layers in the architecture can receive a reasonable gradient and weight update. Then, the normalised output is fed through a linear feed-forward layer with a non-linear activation function, which forms the output of the encoder. Since multiple encoder layers are stacked on top of each other, this process continues, until a final output representation is generated. Repeatedly encoding the input enables a rich and diverse contextual understanding to be formed, as each encoder in the stack can attend to, and learn, representations of different parts of the input.

The decoder stack essentially mirrors the encoder stack, but not only accepts the output of the encoders as input, it also accepts the decoded token at the previous time step. The final component of the decoder is a linear layer, followed by a softmax function, which forms a probability distribution over the next token. The decoder is repeatedly applied until it outputs a special token, which denotes the end of its output. The architecture is trained via standard back-propagation, comparing the predicted softmax distribution to a one-hot encoded ground-truth representation. The encoder learns a highly complex semantic embedding space to represent the current and historical state of the input data, and the decoder is able to generate the likely next token.

Whilst the original transformer demonstrated attention can be used in-place of recurrence, recent work aims to introduce recurrence back into the transformer architecture, with recurrent transformers (Hutchins et al., 2022). This is because the computational complexity of the self-attention mechanism is quadratic with respect to the sequence length, which limits scalability. The idea is to apply the transformer recurrently, over a block of tokens, which enables the architecture to process longer sequences. Instead of using gates like the LSTM, (Hutchins et al., 2022) use another transformer layer to compute the recurrence function. This achieves significant performance benefits compared to the LSTM, since the recurrence function can be parallelised on GPU hardware. Also, the problem of vanishing gradients is mitigated, because the number of recurrent steps are much smaller, compared to applying recurrence on each input token.

The transformer architecture introduced so far, including the recent recurrent transformer, is restricted to accepting sequential input data such as natural language. This architecture has been extended to support visual inputs using the vision transformer (Dosovitskiy et al., 2020), which is of particular relevance to this thesis, since we consider neuro-symbolic learning tasks that involve perception and reasoning. Let us now explain the architecture of the vision transformer.

Vision Transformer The standard transformer accepts a 1-D sequence as input, whereas in the vision case, the input is 2-D in the form of imagery. To tackle this problem, the vision transformer cuts the image data into a sequence of smaller patches, which is flattened into a vector. The flattened image patches are then passed through an embedding layer, along with a positional encoding, to generate the input to the encoder stack. The vision transformer is designed to solve image classification tasks, so a decoder stack is not required, since the output is a classification, rather than the next token in a sequence. To achieve this, a learnable class embedding is prepended to the input sequence of patches, and the value of the class layer following encoding (and self-attention to the image patches), is fed to a MLP to perform the final classification. The vision transformer has demonstrated state-of-the-art performance compared to the CNN. Interestingly, (Dosovitskiy et al., 2020) also suggest using feature maps from a CNN, instead of the raw image data, to create the input patches. This could be used to leverage

a pre-trained CNN that is already capable of extracting low-level image features.

In this thesis, we utilise both the recurrent transformer and vision transformer architectures within baseline approaches in Chapter 6. In practice, both architectures are highly complex, with many possible variants, and tunable components. We encourage the reader to explore the relevant literature if more details are desired (Vaswani et al., 2017; Dosovitskiy et al., 2020; Hutchins et al., 2022). These architectures have inspired recent work in the area of large foundation models, which are trained with vast (internet-scale) quantities of data, and have fuelled significant interest in AI in the last few years. Let us now provide an overview of these architectures.

2.1.3 Foundation Models

The term foundation models refers to deep transformer architectures that are trained on significantly large datasets (Bommasani et al., 2021). The idea is to use such a large pre-trained model as a foundation for performing downstream tasks. With a model that contains such a vast amount of implicit world knowledge, embedded within its latent space, the effort required to adapt the architecture to new tasks is minimal, compared to training a bespoke model from scratch. Often, this effort is as minimal as providing sufficient context to the model in the input sequence during inference, referred to as *prompt tuning*, or *prompt engineering*, to configure the model to solve the task at hand. More complex tasks may require fine-tuning of the model weights, and in these cases, strong performance can be achieved with significantly smaller datasets than what would be required when training from scratch. Foundation models, despite only appearing in the last few years, have had profound implications across AI, and society in general, with many governments, companies, and academic institutions rushing to understand their capabilities, and risks. The first foundation models to appear were based on language (Devlin et al., 2018; Brown et al., 2020), and are often referred to as LLMs, but they have since been extended to support visual inputs (Radford et al., 2021; Li et al., 2022b). Let us now describe how these architectures are trained.

Large Language Models LLMs are trained in three steps:

1. **Pre-training.** Using a vast quantity of data, a transformer is trained to predict the next token in an input sentence. A large training set can be generated automatically, by masking tokens. The masked token forms the ground-truth target word at a given time step, and the previous tokens form the input. This can be considered a form of self-supervised learning.
2. **Instruction tuning.** After the pre-training in step 1, the model has already learned a large amount of world knowledge. However, since it has been trained in a self-supervised fashion to simply predict the next token, and therefore complete sentences, the model is not particularly useful. In the second training step, the model is trained to follow human-generated instructions. The same next token prediction regime is followed, but this time over a high quality hand-curated dataset of instructions and ground-truth answers.
3. **Reinforcement learning from human feedback.** The model is then further refined in a third step that involves aligning the output of the model more closely to human preferences. This step involves training a separate neural network, called a reward model, with data obtained from human preference annotations. Given a set of candidate output generations for a given input, humans rank the outputs in order of their preference. Once the reward model has been trained, it can be used automatically within a reinforcement

learning loop, where the transformer is fine-tuned to predict answers that maximise the reward output by the reward model.

These steps result in a learned transformer that can follow instructions, and output sentences that are more aligned to human preferences. In Chapter 6, we leverage the GPT-4 LLM (OpenAI et al., 2023) to automate the engineering of some of the components within our proposed neuro-symbolic architecture. The result is the user only has to construct a natural language prompt describing the engineering component, and the LLM can generate the implementation.

Vision-Language Models Vision-language models learn to encode joint representations of textual and image data. The vision part of the model learns to recognise spatial features from image data, and the language model encodes information from text. The representations of both modalities is combined within an embedding space, such that if an image of a bird is observed, the model learns to associate this with relevant words. The challenges are: (1) How to perform self-supervised training with image data, and (2) how to align the modalities. The CLIP foundation model solves both of these challenges, by performing joint training of an image encoder, and a text encoder, using a large dataset of captioned images (Radford et al., 2021). The objective is to minimise the *contrastive loss*, which ensures positive data points have small cosine distance in the shared embedding space, and negative data points have large cosine distance. During inference, the text encoder can be used to create a zero-shot classifier, by creating an embedding of the names of the target classes, and the class that has the largest dot product given an encoding of the unseen test image passed through the image encoder is returned. The image encoder can be any model capable of embedding imagery, although a vision transformer is often used. The text encoder is often a standard transformer.

In this thesis, we adopt the BLIP approach in Chapter 6, which is an extension of the CLIP model (Li et al., 2022b). BLIP also uses a multi-modal mixture of encoder-decoder transformer modules. The image encoder is a Vision Transformer, and the text encoder is a BERT model. BLIP is pre-trained to jointly optimise three objectives; (1) *Image-Text Contrastive Loss*, which aligns the embedding space of the vision transformer and BERT by encouraging positive image-text pairs to have similar latent representations in contrast to negative pairs, in a similar fashion to CLIP. (2) *Image-Text Matching Loss*, which is a binary classification task that trains BLIP to predict whether an image-text pair is matched (positive) or unmatched (negative). (3) *Language Modelling Loss* which aims to generate a textual description given an image. This is achieved using the cross-entropy loss with a ground-truth description. BLIP also introduces a caption filtering method, to remove noisy captions present in internet sourced datasets. BLIP is capable of solving a range of tasks, including image-text retrieval, image captioning, VQA, natural language visual reasoning, and visual dialog. In Chapter 6 of this thesis, we utilise the VQA mode, and explore how this can be integrated within a neuro-symbolic architecture to enable symbolic learning from raw data. BLIP can be fine-tuned in this mode by encoding an image-question pair into multi-modal embeddings, before passing the embeddings to a text decoder to obtain the answer. The VQA architecture is then trained end-to-end using ground-truth answers.

2.1.4 Uncertainty

In Chapter 3, we leverage pre-trained CNNs to extract primitive concepts from raw data, where the neural network confidence score is used to calculate example weight penalties for a symbolic

learner. We now present a brief overview of how uncertainty is formulated within neural networks, as well as the state-of-the-art uncertainty aware neural network we leverage in Chapter 3.

Uncertainty in machine learning can arise from multiple sources, and accurately estimating it is not trivial (Hüllermeier and Waegeman, 2021). For example, there may be uncertainty related to the set of training data, in whether it accurately and concretely represents the underlying distribution. Also, there may be uncertainty in how well the chosen architecture is able to generalise from the training data, and whether another architecture would perform better. However, the most common form of uncertainty is predictive uncertainty during inference. In the case of classification tasks, uncertainty refers to which class a new unseen test data point belongs to, if any at all. Uncertainty can be formulated as either *aleatoric* or *epistemic* uncertainty, which generally corresponds to irreducible and reducible uncertainty respectively (Hüllermeier and Waegeman, 2021; Pearce et al., 2021). In a machine learning classification task, aleatoric uncertainty can be thought of as the uncertainty along the class decision boundary, whereas epistemic uncertainty can be thought of as whether the data point falls into any of the classes at all (Hüllermeier and Waegeman, 2021). Epistemic uncertainty is said to be reducible, since one can collect more data, whereas aleatoric uncertainty reflects the inherent uncertainty in making the decision, regardless of the amount of data you have.

The confidence estimates output by a neural network softmax function in a classification task often only capture aleatoric uncertainty. Recall the softmax function from Equation 2.5, where $\mathbf{x}$ represents a neural network output logit, and k is the number of classes:

$$\text{softmax}(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{j=1}^k e^{\mathbf{x}_j}}$$

Softmax outputs are based on a single probability distribution, where neural network output logits are squashed into real values between 0 and 1. There are three challenges with the softmax function in terms of uncertainty quantification. First, the exponent applied to neural network outputs inflates the confidence estimates. Second, as the softmax output is a point-wise, multinomial distribution, it is only possible to compare the confidence of the predicted class among the class decision boundary (representing aleatoric uncertainty), as opposed to estimating the predictive distribution variance to reflect epistemic uncertainty. Third, when softmax is paired with the commonly used cross-entropy loss, the neural network is only trained to minimise prediction error, as opposed to expressing uncertainty robustly. To address these challenges, many techniques have been proposed in the literature, to explicitly encode uncertainty information during neural network training (Rasmussen, 2003; Mackay, 1995; Blundell et al., 2015; Abdar et al., 2021). In Chapter 3 of this thesis, we leverage the Evidential Deep learning framework (Sensoy et al., 2018, 2020) to quantify the uncertainty of neural network predictions, since this approach has achieved state-of-the-art performance on uncertainty quantification tasks.

Assuming a classification task, (Sensoy et al., 2018) replaces the softmax layer in a neural network with a linear layer that represents the parameters of a Dirichlet distribution (Kotz et al., 2019). A Dirichlet distribution is a second-order distribution that inherently models the variance of the predictive distribution, as opposed to the point-wise estimate that softmax outputs. (Sensoy et al., 2018) models the Dirichlet parameters using subjective logic (Jøsang, 2016), which is based upon the Dempster-Shafer Theory of Evidence (DST) — a mathematical framework for modelling epistemic uncertainty (Dempster, 1968). The parameters of the Dirichlet correspond to the possible class labels $j = 1, \dots, k$, and are computed by assigning an amount of evidence e_j to each class label. In (Sensoy et al., 2018), the evidence is the value of the neural network output logit for each class, following a ReLU function to ensure $e_j \geq 0$. The parameters of

the Dirichlet are then computed as $\boldsymbol{\alpha} = \langle e_1 + 1, \dots, e_k + 1 \rangle$, since the Dirichlet is only defined for parameter values > 0 . Also, as the Dirichlet is a second-order distribution, it represents a probability density function of a discrete class assignment, in the form of a probability mass function $\mathbf{p}$. It is characterised as:

$$D(\mathbf{p}|\boldsymbol{\alpha}) = \begin{cases} \frac{1}{B(\boldsymbol{\alpha})} \prod_{j=1}^k p_j^{\alpha_j-1} & \text{for } \mathbf{p} \in \mathcal{S}_k \\ 0 & \text{otherwise} \end{cases} \quad (2.23)$$

where $\mathcal{S}_k$ is the set of all probability mass functions $\mathbf{p}$, i.e., $\mathcal{S}_k = \{\mathbf{p} \mid \sum_{p_j \in \mathbf{p}} p_j = 1 \text{ and } 0 \leq p_1, \dots, p_k \leq 1\}$, and $B(\boldsymbol{\alpha})$ is the k -dimensional multinomial beta function (Johnson et al., 1972). A Dirichlet with all parameters equal to 1, i.e., $D(\mathbf{p}|\langle 1, \dots, 1 \rangle)$ represents a uniform distribution, and total uncertainty for the classification. If more evidence is observed for a particular class sample, the corresponding parameter value is increased, and the probability of that class increases. Since there is a greater amount of evidence across all classes, the overall uncertainty reduces. The expected probability of a particular class sample $\hat{p}_j$ is computed as the mean of the Dirichlet:

$$\hat{p}_j = \frac{\alpha_j}{S} \quad (2.24)$$

and the variance is computed as:

$$Var(p_j) = \frac{\alpha_j(S - \alpha_j)}{S^2(S + 1)} \quad (2.25)$$

where $S = \sum_{j=1}^k \alpha_j$ is known as the “strength” of the distribution, and corresponds to the total amount of evidence. As more total evidence is observed, the denominator in Equation 2.25 becomes larger, the variance decreases, and so does the uncertainty. Compared to softmax, this leads to more robust confidence estimates for neural network predictions. For example, consider the following neural network output logits for a 5-class classification task:

$$\mathbf{x} = \langle 20, 10, 5, 15, 17 \rangle$$

Using the softmax function, the resulting probability output, to 4 decimal places, is:

$$\text{softmax}(\mathbf{x}) = \langle 0.9465, 0.0000, 0.0000, 0.0064, 0.0471 \rangle$$

whereas treating $\mathbf{x}$ as the evidence for each class sample, the parameters of a Dirichlet distribution are $\boldsymbol{\alpha} = \langle 21, 11, 6, 16, 18 \rangle$, and the resulting probability is:

$$\hat{\mathbf{p}} = \langle 0.2917, 0.1528, 0.0833, 0.2222, 0.2500 \rangle$$

It is clear the softmax function has over-inflated the probability estimates, as it assigns a large probability to the first class sample, even though the first logit value is only slightly higher than the last two logit values. Since the probability output from the Dirichlet is computed as the evidence for a parameter divided by the total evidence, the resulting probability values reflect the input logits more closely. The uncertainty for this example is:

$$Var(\hat{\mathbf{p}}) = \langle 0.0028, 0.0018, 0.0010, 0.0024, 0.0026 \rangle$$

Note that if we increase the amount of evidence by multiplying each value of $\mathbf{x}$ by a constant amount, e.g., 5, the predicted probability of each class remains the same, but the uncertainty

decreases:

$$\text{Var}(\hat{\mathbf{p}}) = \langle 0.0005723, 0.0003586, 0.0002116, 0.0004788, 0.0005194 \rangle$$

Whilst (Sensoy et al., 2018) frame the predicted confidence score of the neural network in terms of subjective logic based upon the DST, the key novelty introduced is a method of training a neural network to *learn* the parameters of the Dirichlet, which effectively encodes the ability to express uncertainty within the training objective. Recall $\mathcal{D}$ is a set of training data points, and θ denotes the neural network parameters. The best performing loss function in (Sensoy et al., 2018) is defined as:

$$\mathcal{L}(\mathcal{D}; \theta) = \sum_{j=1}^k \left(\frac{y_{ij} - \alpha_{ij}}{S_i} \right)^2 + \frac{\alpha_{ij}(S_i - \alpha_{ij})}{S_i^2(S_i + 1)} \quad (2.26)$$

where each summand corresponds to the prediction error, and the variance respectively. Since this loss is minimised, the network learns to express uncertainty in a robust manner. In contrast, training a network with the softmax function paired with the cross-entropy loss, only minimises the prediction error. In Chapter 3 of this thesis, we use an extension of the evidential deep learning framework, called EDL-GEN (Sensoy et al., 2020), which treats the output of each class as a binary decision problem, and computes the Dirichlet parameters as a density estimate, using in and out of distribution data sets. The out of distribution data contains samples that do not belong to any of the k classes. The log output of a binary classifier is used to predict whether an input sample belongs to the in or out of distribution data, which is used as the density estimate. Evidence is calculated by applying an exponential to the logit output, and the Dirichlet parameters are computed in the usual manner. Instead of using the loss function presented in Equation 2.26, the EDL-GEN network is trained using a Bernoulli loss, which trains the network to distinguish between in and out of distribution samples. The out of distribution data is generated automatically using a Variational Autoencoder, which improves upon alternative approaches that require hand-tuned data sets. For more details, we refer the reader to (Sensoy et al., 2020).

2.2 Symbolic AI

Instead of learning over continuous vector representations, symbolic AI represents knowledge as discrete symbols. This has numerous advantages over deep learning. First, the knowledge is inherently interpretable, since the symbols often correspond directly to real-world concepts. In deep learning, knowledge is expressed in the form of weights and biases within a neural network, which is challenging to interpret. Second, a symbolic representation enables the use of formal methods that guarantee correctness and optimality, such as search, planning, program induction, and reasoning. Deep learning on the other hand, *approximates* functions, and is not guaranteed to converge to an optimal solution unless the loss function is convex. Finally, since symbolic AI represents knowledge explicitly, often only a few examples are required to perform program induction and learn general knowledge, whereas deep learning requires many examples to generalise appropriately over the desired data distribution. In this section, we present the fundamental concepts within symbolic AI, building towards the LAS framework this thesis leverages, for learning general, interpretable, and expressive knowledge from raw data.

2.2.1 Computational Logic

The fundamental building block of symbolic AI is computational logic (Huth and Ryan, 2004; Paulson, 2018), since logic defines a formal framework for reasoning over symbols, representing what is true, and what is false, enabling us to prove certain properties about the world. Let us now introduce the basic form of logic, called *Propositional Logic*, followed by its extension to *First-Order Logic*, and *Fuzzy Logic*. The text in this sub-section largely follows (Huth and Ryan, 2004; Cingillioglu, 2022).

Propositional Logic Propositional Logic enables us to represent propositions, or declarative statements, which one can argue as being true or false. For example:

- The number 4 is even.
- It is snowing.
- An alien landed in the garden.
- I won the lottery yesterday.
- I bought a lottery ticket.
- I won the raffle at the village fair.

Each of these sentences can be assigned a truth value. Propositional Logic assumes these sentences are atomic and not decomposable. The symbols $p, q, r, \dots$ called *propositional atoms*, are assigned to each atomic statement, to enable reasoning over more complex sentences in a compositional manner. There are also various *connectives* which enable complex sentences to be formed:

- **Negation** ($\neg$): The negation of p is expressed as $\neg p$ and denotes the logical inverse of a proposition. For example, if $p = \text{"It is snowing"}$, then $\neg p = \text{"It is **not** snowing"}$.
- **Disjunction** ($\vee$): Disjunction is used to represent logical OR, which, given a pair of propositions states at least one proposition is true. For example, if $p = \text{"I won the lottery yesterday"}$, and $q = \text{"I won the raffle at the village fair"}$, $p \vee q$ states you either won the lottery yesterday, or the raffle at the village fair.
- **Conjunction** ($\wedge$): Conjunction is used to represent logical AND, which, given a pair of propositions states both are true. This is represented as $p \wedge q$.
- **Implication** ($\rightarrow$): Implication enables us to express that a proposition q is the logical consequence of p . For example, if $q = \text{"I won the lottery yesterday"}$, and $p = \text{"I bought a lottery ticket"}$, we can write $p \rightarrow q$. p is called the assumption, or premise, and q is called the conclusion.
- **Equivalence** ($\leftrightarrow$): Equivalence is also referred to as the double implication, and asserts that if p is true, then q is true, and if q is true, then p is true. It means that p and q are true under exactly the same circumstances.

These connectives can then be used repeatedly to form a more complex propositions, called a *formula*, such as:

$$p \wedge q \rightarrow \neg r \vee q$$

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\neg p \vee q$
0	0	1	0	0	1	1	1
0	1	1	0	1	1	0	1
1	0	0	0	1	0	0	0
1	1	0	1	1	1	1	1

Table 2.1: Truth table for the propositional connectives, where $\top = 1$ and $\perp = 0$

which means that if p **and** q are true, then **not** r **or** q is true. To avoid ambiguity between the order of application of the connectives, parentheses can be used. In the case no parentheses are provided, a standard order is assumed: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. There are two other symbols defined in propositional logic: $\top$ (“top”), and $\perp$ (“bottom”), which denote true and false respectively. So far we have defined the *syntax* of propositional logic, let us now define the *semantics*, which associates meaning to each of the symbols.

A *model* of a formula is an assignment of each propositional atom to a truth value. This is often represented by a truth table that enumerates all possible models. Assuming two propositions p and q , the truth table for each connective is shown in Table 2.1. The semantics of negation, conjunction, and disjunction are intuitive, negation flips the truth value of the proposition, conjunction is only true if both propositions are true, and disjunction is true if either (or both) of the propositions are true. The semantics of implication is less intuitive. This can be thought of as checking whether truth is being preserved. If p and q are true, then clearly, the implication holds, whereas if p is true and q is false, the implication does not hold. In the cases where p is not true, it does not matter what the truth value of q is, as there is no truth to be preserved. Therefore, the implication is true in these cases. To make this more explicit, we have included the truth values for the formula $\neg p \vee q$ in Table 2.1. Notice how this is semantically equivalent to implication, since the values in the truth table match. This is denoted $p \rightarrow q \equiv \neg p \vee q$.

A formula that always evaluates to true (i.e., in every model) is called a *tautology*, and a formula that always evaluates to false is called a *contradiction*. For example, $p \vee \neg p$ is a tautology, and $p \wedge \neg p$ is a contradiction. A formula is said to be *satisfiable*, if there is at least one model in which the formula evaluates to true. A formula A is said to *entail* another formula B , if under all models in which A is true, B is also true. This is denoted $A \models B$. For example, $p \wedge q \models p \vee q$. Note that $p \vee q$ does *not* entail $p \wedge q$. This is denoted $p \vee q \not\models p \wedge q$. Finally, given a set of formulas $A_1, \dots, A_n$, and another formula B , an *argument* of the form $A_1, \dots, A_n \models B$ is said to be *valid*, if in every situation $A_1, \dots, A_n$ are true, B is also true. In this case, $A_1, \dots, A_n$ and B are called the premises, and conclusion, respectively. A common example is *modus ponens*, which can be used to eliminate implication. Modus ponens states that $A, A \rightarrow B \models B$ is a valid argument, because in every model where A , and $A \rightarrow B$ are true, the conclusion B is also true. The difference between $\models$ and $\rightarrow$, is that $\rightarrow$ is defined over individual atoms, whereas $\models$ is defined over all possible models of the given formulas.

A proof system contains a collection of rules, or procedures, that enable us to infer a conclusion, given a set of premises. This enables one to reason and make deductions, without evaluating every possible situation in which a set of formulas can be true (i.e., it avoids enumerating the truth table, which has 2^n rows, where n is the number of atoms). A proof system is a purely syntactic way of reasoning about what the symbols mean. We write $A_1, \dots, A_n \vdash B$ if a proof system concludes a formula B , given formulas $A_1, \dots, A_n$. Note that $\vdash$ is the syntactic counterpart of $\models$, and simply means we can derive the conclusion by following the rules in the proof system. Natural deduction is a popular proof system for propositional logic, and includes

the modus ponens rule, as well as rules which closely follow the connectives. An example is the conjunction rule, which enables us to conclude $A \wedge B$, if we have already concluded A and B separately. Ideally, if the syntactic proof system concludes a certain formula, we would like this to correspond to semantic entailment. This correspondence is denoted by whether the proof system is *sound* and *complete*. Given formulas $A_1, \dots, A_n$, and B , a proof system is:

- **Sound**, if $A_1, \dots, A_n \vdash B$ implies $A_1, \dots, A_n \models B$. This means that if $A_1, \dots, A_n \vdash B$ is proven by the proof system, then B is true in all models where $A_1, \dots, A_n$ are true, according to the semantics of propositional logic. In other words, a proof system is sound if it never derives false conclusions from true premises.
- **Complete**, if $A_1, \dots, A_n \models B$ implies $A_1, \dots, A_n \vdash B$. This means that every statement that is true according to the semantics of propositional logic can be proven using the proof system.

These properties essentially ensure that everything that can be proved is true, and everything that is true can be proven. The natural deduction proof system, often used in propositional logic, is sound and complete (Huth and Ryan, 2004).

Propositional logic is limited to reasoning over declarative statements that can be assigned a definitive truth value. In reality, declarative statements can be more complex, and include quantifiers such as “all”, and “there exists”, which apply scoping to the various propositions. For example, consider the sentence “Every day it is sunny, there is an ice cream van outside”. In propositional logic, one might be tempted to represent this using an implication, but this does not capture the finer structure of the sentence, i.e., the phrases “every”, and “there is an”. A more expressive representation is therefore required.

First-Order Logic First-order logic extends propositional logic by introducing predicates that denote Boolean relations over variables, constants and function symbols, which are collectively called *terms*. For example, in the sentence “In every forest, if there is a redwood tree, this casts more shade over the other trees in the forest”, we can define the predicates $Forest(f)$, $Redwood(t, f)$, and $OtherTree(t', f)$ to denote that f is a forest, t is a redwood tree in f , and t' is another tree in f that is not a redwood, respectively. We can also define the predicate $MoreShade(t, t')$ to denote the relation that t casts more shade over t' . Here the lowercase letters are called *variables*, which enable us to express more general statements. Without variables, it would be tedious to use these predicates for every combination of t and t' . At this stage, we are still missing some key components of the sentence, such as “in every forest”, and “if there is”. This is where quantifiers come in. They are defined as $\forall$, meaning “for all”, and $\exists$, meaning “there exists”. Putting all of these building blocks together, we can now express our sentence in first-order logic:

$$\forall f (Forest(f) \wedge \exists t Redwood(t, f) \rightarrow \forall t' (OtherTree(t', f) \rightarrow MoreShade(t, t')))$$

which should be read as “For all f , if f is a forest and there exists a t such that t is a redwood tree in f , then for all t' , if t' is another tree in f that is not a redwood, then t casts more shade than t' ”. Recall that “is not a redwood” was assumed in our meaning of the $OtherTree$ predicate. In this example, the predicate $Forest$ is a *unary predicate*, since it accepts one argument, whereas the other predicates are *binary predicates*, since they accept two arguments. The number of arguments is also called the *arity*, and predicates with any finite arity are supported in first-

order logic. Formally, the syntax of first-order logic extends that of propositional logic with the addition of the following symbols:

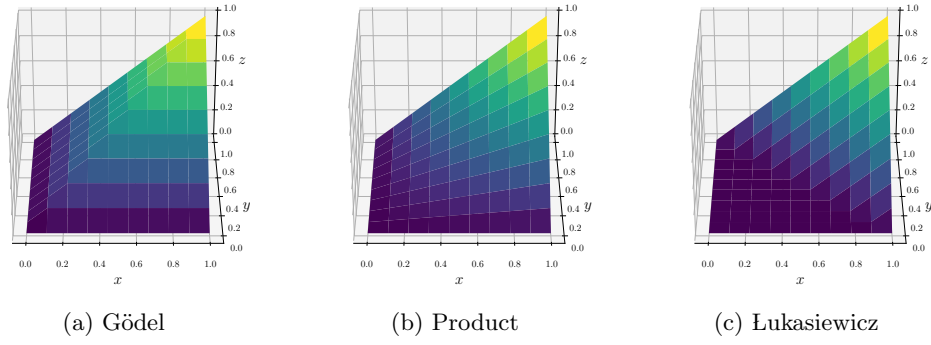
- **Constants** denote individual *objects* using lowercase letters such as a, b, c, a_1 . In first-order logic, objects correspond to real-world entities that concerns the task at hand.
- **Variables** denote placeholders for objects. These are also denoted by lowercase letters such as $x, y, \dots$, but are quantified using a quantifier.
- **Function symbols** such as f, g , denote an n -ary mapping from a constant to another constant.
- **Predicate symbols** denote Boolean relations among entities. They start with capital letters.
- **Quantifiers** are defined as $\forall$ (for all) and $\exists$ (there exists).

Constant, function, and predicate symbols collectively form a *signature*. Variables are excluded since they merely act as syntactic placeholders, and don't have any meaning of their own. A formula in first-order logic with no variables is said to be *ground*, and a ground formula is equivalent to a propositional formula, since every ground term can be seen as a propositional atom. If a variable is quantified, it is said to be *bound* by the quantifier, otherwise it is *free*. Quantifiers have the same binding strength as negation, and are applied before other connectives. Function symbols enable a more compact representation for certain predicates. For example, consider the predicate $Mother(x, y)$, which denotes x is the mother of y . We can express this using a function symbol $m(y)$, which returns x , the mother of y . This has the advantage that we can remove a variable (and it is quantifier), simplifying the representation. Now let us define the semantics of first-order logic.

The notion of a model is extended from propositional logic as follows. Given a signature, a *model* is as a tuple $\mathcal{M} = \langle \mathcal{D}, \mathcal{I} \rangle$, where $\mathcal{D}$ is a non-empty domain of objects, and $\mathcal{I}$ is an interpretation function. $\mathcal{D}$ represents a (potentially infinite) set of objects, and $\mathcal{I}$ assigns a meaning, or an *interpretation* to each symbol in the signature, as follows:

- If a is a constant symbol, then $\mathcal{I}(a)$ is an object in $\mathcal{D}$.
- If f is a function symbol of arity n , then $\mathcal{I}(f)$ is a function that maps n objects in $\mathcal{D}$ to another object in $\mathcal{D}$.
- If P is a predicate symbol of arity n , then $\mathcal{I}(P)$ is a subset of $\mathcal{D}^n$, representing a relation between objects in $\mathcal{D}$.

Recall that variables have no inherent semantics, so they are excluded from an interpretation. However, in order to define semantic entailment, we require an assignment of variables to objects in the domain. If x is a variable, an assignment is defined as a mapping $\mathcal{V}(x) \in \mathcal{D}$. This is often referred to as *grounding*, since if every variable is assigned a value, the formula becomes ground. A model $\mathcal{M}$, and an assignment $\mathcal{V}$ is said to entail a formula A , if A evaluates to true, given $\mathcal{M}$ and $\mathcal{V}$. This is expressed as $\mathcal{M}, \mathcal{V} \models A$. If A is true for *every* assignment of the variable x , given $\mathcal{M}$, we write $\mathcal{M} \models \forall x A$, and we write $\mathcal{M} \models \exists x A$ if A is true for *at least one* assignment. If A is true in every assignment, A is said to be *valid*, whereas if A is true in at least one assignment, it is said to be *satisfiable*. As for a proof system, natural deduction can be extended to handle quantifiers, and is sound and complete under these extensions (Huth and Ryan, 2004). However,

Figure 2.3: Popular t -norms that model conjunction in fuzzy logic.

unlike propositional logic, the problem of checking validity is undecidable. This is due to a potentially infinite domain of objects which can be universally quantified, and having to check all possible models.

Fuzzy Logic Classical Boolean logic, such as propositional and first-order logic as introduced thus far, is restricted to representing binary conclusions that are either true or false. In reality, propositions may be true with a varying degree of certainty. *Fuzzy logic* relaxes the hard binary decision of classical logic, and assumes the truth value can be a real number in $[0, 1]$ (Metcalf et al., 2008; Novák et al., 2012). In doing so, one has to consider how to combine propositions together with varying degrees of truth, and when combined, what the overall degree of truth should be for a certain formula. To achieve this, triangular norms, often called t -norms, can be used (Gupta and Qi, 1991; Klement et al., 2013). A t -norm is a binary operation $t : [0, 1]^2 \rightarrow [0, 1]$ that satisfies the following properties, $\forall x, y, z \in [0, 1]$:

1. **Associativity:** $t(t(x, y), z) = t(x, (t(y, z)))$.
2. **Commutativity:** $t(x, y) = t(y, x)$.
3. **Monotonicity:** $t(x, z) \leq t(y, z)$, where $x \leq y$.
4. **Boundary Condition:** $t(1, x) = x$.

There are many possible t -norms that can be used (Klement et al., 2013). In fuzzy logic, t -norms correspond to a particular logical connective. For example there are three popular t -norms that model conjunction:

- **Gödel:** $t(x, y) = \min(x, y)$
- **Product:** $t(x, y) = x \cdot y$
- **Łukasiewicz:** $t(x, y) = \max(0, x + y - 1)$

The surface of these t -norms is presented in Figure 2.3, where the x and y axes are the inputs and the z axis is the output. The magnitude of the z surface decreases from left-to-right, and the Gödel t -norm has the largest z surface. In Chapter 3, we leverage t -norms to aggregate the confidence scores from neural network predictions, to define an appropriate weight for the symbolic learner. The t -norms introduced so far are defined over two inputs, whereas we require the t -norm to generalise over n inputs. A t -norm can be generalised using the associativity

property. Let $t : [0, 1]^2 \rightarrow [0, 1]$ be a t -norm. A generalised t -norm, $t_n : [0, 1]^n \rightarrow [0, 1]$, where $n \geq 2$, is recursively defined as follows:

$$\begin{aligned} t_2(x_1, x_2) &= t(x_1, x_2) && \text{(base case)} \\ t_n(x_1, \dots, x_n) &= t(t_j(x_1, \dots, x_n), t_{n-j}(x_{j+1}, \dots, x_n)) && \text{(inductive case)} \end{aligned}$$

where the associativity property of t ensures the inductive case is the same for all $1 \leq j \leq n$.

2.2.2 Answer Set Programming

In this thesis, we focus on learning symbolic knowledge from raw data, where the symbolic knowledge is expressed as an ASP program. We now present an overview of ASP (Gelfond and Lifschitz, 1988), to enable the reader to grasp the contributions presented in Chapters 3–6. This text is largely based upon the material in (Gelfond and Kahl, 2014; Law, 2018; Lifschitz, 2019; Furelos-Blanco, 2023).

ASP is a *declarative* form of programming, where the developer encodes a representation of the problem itself, rather than a recipe, or algorithm, to procedurally solve the problem. The latter is the common approach with *imperative* programming languages (e.g., Python, Java, C++, etc.). To encode a representation of the problem declaratively, one can express conditions on the values of variables that describe solutions to the original problem. An automated reasoning system then takes this “specification” as input, and computes the solutions. For certain classes of problems, such as combinatorial optimisation problems, expressing a program in a declarative manner is often significantly more natural than an imperative counterpart. For example, a general solution to the Hitting Set problem (Karp, 1972) presented in Chapter 4, can be expressed in four lines of ASP, whereas the imperative counterpart would be significantly more complex, and potentially difficult to generalise in full. ASP is based upon the field of Logic Programming (Lloyd, 2012), which extends the mathematical formalisation of first-order logic to the programming domain. The syntax of ASP is as follows.

An *atom* is an expression of the form $p(t_1, \dots, t_n)$, where p is a predicate symbol of arity n , and $t_1, \dots, t_n$ are terms. If there are no terms, i.e., $n = 0$, then no parentheses are used. The definition of a term is inherited from first-order logic, where terms include constants, variables, and function symbols. In this thesis, a term is either a constant, or a variable. In ASP, variables are expressed using capital letters ($X, Y, \dots$), whereas constants are expressed using lowercase letters ($a, b, \dots$). In a similar fashion to first-order logic, a *ground* atom contains no variables. A *literal* is defined as an atom a or its negation $\text{not } a$, where not represents *negation as failure* (Clark, 1977) (i.e., if a cannot be proved, it is assumed to be false). In this thesis, we assume an ASP program includes three types of rules:⁵

- **Normal rule.** A normal rule is of the form $h : - b_1, \dots, b_n$, where h is an atom called the *head* of the rule, and $b_1, \dots, b_n$ are literals called the *body* of the rule. The symbols “ $: -$ ” and “ $,$ ” represent implication and conjunction, respectively. The rule should be read as “ h is true if $b_1 \dots$ and $\dots b_n$ are true”. A normal rule with an empty body is called a *fact*. Rules containing only positive body literals, are known as *definite* rules.
- **Choice rule.** A choice rule is of the form $x\{h_1, \dots, h_m\}y : - b_1, \dots, b_n$, where x and y are called the lower and upper bound, respectively. This means a minimum x , and a maximum

⁵Other rules include weak constraints, aggregates, and optimisation statements, which are outside the scope of this thesis. See (Lifschitz, 2019) for an overview.

y of head atoms are “chosen” from $\{h_1, \dots, h_m\}$. This choice occurs if $b_1, \dots, b_n$ are true. Choice rules are used to represent disjunction, since an ASP solver will generate all possible choices.

- **Constraint.** A constraint is a normal rule with an empty head, i.e., $:- b_1, \dots, b_n$. This can be read as “Reject if $b_1, \dots, b_n$ are true”.

Now let us present the semantics of ASP, which is based upon the stable model semantics (Gelfond and Lifschitz, 1988). Stable models, or *answer sets*, represent solutions to the original problem. Answer sets are obtained by solving an ASP program, and contain only ground literals. In this thesis, we rely upon the popular Clingo ASP solver (Gebser et al., 2019). Let us now outline the mechanics behind solving ASP programs. The *Herbrand base* of an ASP program P is the set of all ground atoms constructed from constants and predicate symbols in P . A *Herbrand interpretation* I of P is a truth assignment of every atom in the Herbrand base. By convention, a Herbrand interpretation denotes the set of atoms which are assigned *true*. A program is *ground*, if every rule in the program can be replaced with a ground instance of the rule. A Herbrand interpretation I satisfies a ground rule r , if for every rule where I satisfies the body of r , the head of r is also satisfied. The body of r is satisfied by I if $body^+(r) \subseteq I$ and $body^-(r) \cap I = \emptyset$, where $body^+(r)$ and $body^-(r)$ denotes the set of positive and negative body literals of r respectively. The head h of r is satisfied by I depending on the type of rule:

- **Normal rule.** If r is a ground normal rule, h is satisfied by I if $h \in I$.
- **Choice rule.** If r is a ground choice rule, h is satisfied by I if the number of satisfied atoms in the head is between the lower and upper bounds, i.e., $a \leq |I \cap \{h_1, \dots, h_m\}| \leq b$.
- **Constraint.** If r is a constraint, since there is no head, r is satisfied if the body of r is satisfied.

A Herbrand interpretation I is called a Herbrand model of a program P , if I satisfies every rule in P . A Herbrand model I is said to be a *minimal* model of P , if there exists no I' s.t. I' is another Herbrand model of P and $I' \subset I$. To simplify the solving process, the *reduct* of P is computed, which can be thought of as a reduction of the program to reduce complexity, by transforming P into a program with only definite rules. Formally, given a ground program P , the reduct of P with respect to a Herbrand interpretation I , denoted P^I , is constructed using four steps (Law, 2018; Furelos-Blanco, 2023):

1. The (empty) heads of all constraints are replaced with $\perp$.
2. For each choice rule r , if the head of r is not satisfied by I , the head is replaced with $\perp$. Otherwise, for each $h \in I$, the rule $h : -body(r)$ is inserted, and finally, r is removed. This effectively unpacks the choice rule and converts it into an equivalent set of normal rules.
3. Any rule r which contains a negated body literal in I is removed, i.e., if $\text{not } a \in body(r)$, and $\text{not } a \in I$, r is removed.
4. Any negative body literals from the remaining rules are removed.

After following these steps, P^I is a ground program which contains only definite rules. For any definite program, there is a *unique* minimal Herbrand model (Gelfond and Lifschitz, 1988). A Herbrand interpretation I is an *answer set* of P if I is the unique minimal Herbrand model of the reduct P^I .

2.2.3 Learning from Answer Sets

Since in this thesis we are interested in *learning* ASP programs, let us now present the LAS framework upon which this thesis is based (Law et al., 2014; Law, 2018). The LAS framework implements ILP (Muggleton, 1991) in the domain of ASP. In ILP, the goal is to learn an inductive hypothesis H , given background relations B , a hypothesis search space of potential rules $\mathcal{H}$, and a set of training examples E . The training examples contain positive E^+ and negative E^- examples, and the goal is to find a $H \subset \mathcal{H}$ such that:

$$\begin{aligned} B, H &\models e, \forall e \in E^+ \\ B, H &\not\models e, \forall e \in E^- \end{aligned}$$

In the case B already contains some relations, the learned H effectively completes the background knowledge B . Otherwise, in cases where B is empty, H represents the complete solution. In order for ILP to be of use in a neuro-symbolic setting, the exact formulation of coverage is highly restrictive. Classical ILP assumes the examples are free of noise, which in a neuro-symbolic case is an unrealistic assumption. The LAS framework not only handles the classical ILP case, but it also provides formal guarantees of optimality in the presence of noisy examples (Law et al., 2018). Examples in the LAS framework are called Weighted Context Dependant Partial Interpretations (WCDPIs), and are defined as tuples $e = \langle e_{\text{id}}, e_{\text{pen}}, e_{\text{pi}}, e_{\text{ctx}} \rangle$, where:

- e_{id} is an example identifier.
- e_{pen} is a positive integer, or ∞ , called a weight penalty.
- $e_{\text{pi}} = \langle e_{\text{inc}}, e_{\text{exc}} \rangle$ is a partial interpretation consisting of a set of inclusion and exclusion atoms, respectively.
- e_{ctx} is a set of contextual atoms.

In the case of positive examples, a hypothesis is said to *cover* the example e , denoted by $H \models e$, if $H \cup B \cup e_{\text{ctx}} \models e_{\text{inc}}$ and $H \cup B \cup e_{\text{ctx}} \not\models e_{\text{exc}}$. In the case of negative examples, the opposite must hold. The role of the weight penalty is to support learning from noisy examples. If the penalty is infinite, this reflects classical ILP, and the example must be covered. Otherwise, the integer penalty is paid if a candidate hypothesis does *not* cover the example. The LAS framework defines an optimisation criteria for searching for a hypothesis that maximises the example coverage in the presence of noise. Formally, the LAS symbolic learners accept as input a set of background knowledge B expressed in the language of ASP, a hypothesis space $\mathcal{H}$ that is often defined by a set of mode declarations,⁶ and a set of WCDPI training examples E . The goal is to learn an inductive hypothesis $H \in \mathcal{H}$ that minimises a scoring function based on the length of H , and its example coverage. Let us assume the set of examples uncovered by H is denoted by $UNCOV(H, (B, E))$. The score of H is given by $score(H, (B, E)) = |H| + \sum_{e \in UNCOV(H, (B, E))} e_{\text{pen}}$. A LAS symbolic learner computes the optimal solution H^* that satisfies:

$$H^* = \arg \min_{H \in \mathcal{H}} [score(H, (B, E))] \quad (2.27)$$

This can be interpreted as jointly maximising the generality of H (i.e., a concise ASP program), and the example coverage. In practice, and as performed in this thesis, e_{inc} and e_{exc} are used

⁶Mode declarations define which predicates should appear in the head and the body of the rules in the search space (Muggleton, 1995).

to encode the labels of the task, and e_{ctx} is used to encode the example features, or concepts. We leverage the ILASP (Law et al., 2020b) and FastLAS (Law et al., 2020a) systems from the LAS framework in Chapters 3–6. ILASP is more expressive than FastLAS, since it supports non-observational predicate learning, where the learned concepts are not directly present in the training examples. FastLAS on the other hand, is more scalable than ILASP, since it computes an optimal subset of the hypothesis space and solves the resulting ASP program in one-shot. In contrast, ILASP considers the full hypothesis space and adopts an iterative algorithm for solving the task. Both LAS systems have achieved state-of-the-art results in performing inductive learning of ASP programs from noisy examples.

Chapter 3

Feed-Forward Neuro-Symbolic Learning

When considering a neuro-symbolic integration, the first question to ask, is whether existing techniques can be combined in a feed-forward fashion. There are a variety of symbolic components available that can learn knowledge, and there are also off-the-shelf pre-trained neural networks, that can extract primitive concepts from raw data.¹ However, there are numerous challenges in such an integration, given the distinct differences between the neural and symbolic paradigms. For example, the symbolic learner requires discrete input, whereas the neural network learns over continuous representations. Also, traditional symbolic learners are vulnerable to noisy inputs, and there is no guarantee pre-trained neural networks will predict accurately for a given task. This is especially the case when new data is observed, that originates from a different distribution than the data used for pre-training. In these situations, known as *distributional shifts*, the neural networks not only are likely to predict incorrectly, but may also predict with high confidence. In this chapter, we investigate the following research questions: (1) Can state-of-the-art symbolic learners, that demonstrate learning from noisy inputs, tolerate noisy neural network predictions and learn accurate knowledge? (2) Once the knowledge is learned, can this knowledge be combined with the neural network predictions during inference? (3) Finally, can the learned knowledge be configured to output probabilistic predictions, given the uncertainty coming from the neural network?

In this chapter, we address these open questions, and present our first contribution that can *learn* symbolic knowledge from raw data, in a feed-forward fashion. Our solution, called FF-NSL, bridges the neural and symbolic components, by transforming the network predictions into a symbolic form, and creating weighted training examples for the symbolic learner, based upon the neural network confidence scores. This effectively automates the generation of the symbolic learning task, provided a suitable search space, and any existing and relevant background relations are given. Through our general framework, we also define the requirements for the symbolic learner, and the neural network. The symbolic learner must support contextual facts within its training examples, in order to capture the neural network predictions of the primitive concepts, and, the examples must be weighted, to capture the level of confidence output from the neural network. We therefore instantiate our framework with the LAS symbolic learners, since they meet both requirements. They have also achieved state-of-the-art results when learning from

¹See <https://modelzoo.co/> and https://pytorch.org/serve/model_zoo.html for examples of pre-trained networks that are widely available as open-source.

noisy examples in a purely symbolic setting (Law et al., 2018). FF-NSL supports both LAS systems, FastLAS, and ILASP, since each provides capability useful in different scenarios. For example, FastLAS is able to scale to large search spaces, which enables a more general search space to be used (Law et al., 2020a). ILASP on the other hand, is able to learn more expressive knowledge such as choice rules, and can also solve tasks that require non-observational predicate learning, where the concepts predicted by the neural network can be linked to the downstream label via an invented predicate (Law et al., 2020b). As for the neural network, FF-NSL requires a probability distribution over the possible values of the concepts to be returned as output. This requirement encompasses a wide range of neural architectures, that can be customised for a given task. Once the knowledge is learned, inference is performed using a logic program that consists of the learned knowledge, any background relations specified, and contextual facts generated from the neural network predictions for unseen raw data. The logic program can be solved to output the final prediction of the task. To extend the inference capability of FF-NSL, we also explore probabilistic inference using a probabilistic programming framework, which enables FF-NSL to return uncertainty information alongside the final output.

To evaluate FF-NSL, we consider the following questions. (1) Can FF-NSL learn accurate knowledge, when an increasing percentage of training data points are subject to distributional shifts? (2) Can an uncertainty-aware neural network help FF-NSL learn more accurate knowledge by providing more robust confidence estimates? (3) During inference, can FF-NSL predict with reasonable probability estimates that reflect the uncertainty coming from the neural network predictions, in the case of increasing percentages of test data subject to distributional shifts? (4) How does FF-NSL compare to other hybrid architectures, where the same neural network is connected to deep learning and statistical baselines?

To address question 1, we introduce new datasets and tasks that require learning complex knowledge from raw data. We also apply various distributional shifts to the raw input, which enables a comprehensive evaluation in more real-world settings. We demonstrate FF-NSL is able to learn accurate knowledge, until $\sim 80\%$ of the training data points are subject to distributional shifts. To address question 2, we instantiate FF-NSL with an uncertainty-aware neural network based upon evidential deep learning (Sensoy et al., 2020), and compare the accuracy of the learned knowledge to the case where a standard neural network is used. Our results indicate the uncertainty-aware network generally provides a small benefit compared to a standard network, and the LAS symbolic learners are able to tolerate noisy confidence estimates. To address question 3, we configure FF-NSL to use the ProbLog system for probabilistic inference (De Raedt et al., 2007), once the knowledge has been learned, and show that the output probability indeed reflects the decreasing accuracy of the final predictions, when an increasing percentage of test data points are subject to distributional shifts. Finally, to address question 4, we compare FF-NSL to hybrid architectures with a neural network and a Random Forest as the symbolic learner. These baselines are chosen because neural networks are powerful architectures capable of scaling to complex tasks, and the Random Forest is considered a somewhat interpretable model, since decision-tree style rules are learned. Our results demonstrate FF-NSL outperforms both baseline models on four tasks, in terms of the accuracy, and interpretability, of the learned knowledge, especially when distributional shifts are applied to the raw inputs. Furthermore, FF-NSL also outperforms these baselines, even when they are trained with significantly more data.

The remainder of this chapter is organised as follows. We present the general FF-NSL framework in Section 3.2. We introduce each task used in our evaluation in Section 3.3, alongside instantiations of the framework with the LAS systems. We then present our evaluation in Sections 3.4–3.6, where we consider the neural network performance under distributional shifts, the

learned hypothesis accuracy, a complete framework evaluation, and, the extension to FF-NSL to enable probabilistic reasoning during inference, respectively. Finally, we summarise the contributions in Section 3.7. Some of the work in this chapter was presented at the 2022 International Joint Conference on Learning and Reasoning, and is based upon the subsequent publication “*FFNSL: Feed-Forward Neural-Symbolic Learner*” in the Machine Learning Journal (Cunnington et al., 2023a).

3.1 Problem Formulation

We consider tasks where the input is a sequence of raw data in the form of images, and the output represents a higher-level prediction that requires reasoning over primitive concepts observed in the input. The knowledge required for reasoning needs to be learned. Note that the final output, given as a label during training, is not represented directly in the input. Therefore, we decompose the task into a perception function $g : \mathcal{X} \rightarrow \mathcal{Z}$, and a reasoning function $h : (\mathcal{Z} \times \mathcal{M})^n \rightarrow \mathcal{Y}$. g maps a raw image $\mathbf{x} \in \mathcal{X}$ into a primitive concept $z \in \mathcal{Z}$, and h maps n pairs of concepts and associated meta-data $m \in \mathcal{M}$ to a target label $y \in \mathcal{Y}$. We assume the target label represents a classification over a set of possible outputs. The reasoning function is defined as $h = H \cup B$, where H is a *hypothesis* that performs the required reasoning, and B is an optional background knowledge that defines a set of predicates and facts relevant to the task. B is assumed to be given, and H needs to be learned. Consider a training set $\mathcal{D}_{\text{task}} = \{\langle X, M, y \rangle, \dots\}$, where $X = \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$ is an n -length sequence of images, $M = \langle m_1, \dots, m_n \rangle$ is an n -length sequence of meta-data associated with X , and $y \in \mathcal{Y}$ is a target label. The goal is to learn g and H , such that:

$$h(\langle (g(\mathbf{x}_1), m_1), \dots, (g(\mathbf{x}_n), m_n) \rangle) = y \text{ for all } \langle X, M, y \rangle \in \mathcal{D}_{\text{task}} \quad (3.1)$$

Intuitively, this means to output correct predictions for all of the training data points, where h outputs the correct target label, given a sequence of pairs of concept predictions from g , and the meta-data. In this chapter, we adopt a sequential approach to learning g and H , where g is learned using primitive concept labels (i.e., ground-truth $z \in \mathcal{Z}$), and H is learned using the predictions from g . The motivation is to investigate the robustness of the symbolic learner when learning general knowledge from noisy concept predictions, since the perception component may not achieve perfect performance, and the input images could be subject to distributional shifts.

Example 3.1.1. The Follow Suit Winner task. *In this task, the goal is to learn the rules of the Follow Suit playing card game: “The winner is the player with the highest ranked card with the same suit as Player 1”. We assume the game is defined over a standard deck of 52 cards. A training data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ contains a sequence of card images, and a sequence of associated meta-data that indicates which player has played each card. The target label is the winning player. Figure 3.1 shows an example data point for a 4-player game. In this task, the neural network is used to extract the concepts from each card image, and the symbolic learner learns the rules of the card game that classify the winner. g returns a concept prediction for each image, which represents the suit and rank values of the card. We assume $g(\mathbf{x}) \in \{1 \dots 13\}$ represents ascending ranks in the “Hearts” suit (where “Ace” is considered high), $g(\mathbf{x}) \in \{14 \dots 26\}$ represents ascending ranks in the “Diamonds” suit, and $g(\mathbf{x}) \in \{27 \dots 39\}$ and $g(\mathbf{x}) \in \{40 \dots 52\}$ represent ascending ranks in the “Spades” and “Clubs” suits respectively. Assuming g predicts correctly, $g(\mathbf{x}_1) = 1$, $g(\mathbf{x}_2) = 21$, $g(\mathbf{x}_3) = 10$, and $g(\mathbf{x}_4) = 52$, representing the “Two of Hearts”, “Nine of Diamonds”, “Jack of Hearts”, and the “Ace of Clubs” respectively. Assuming*

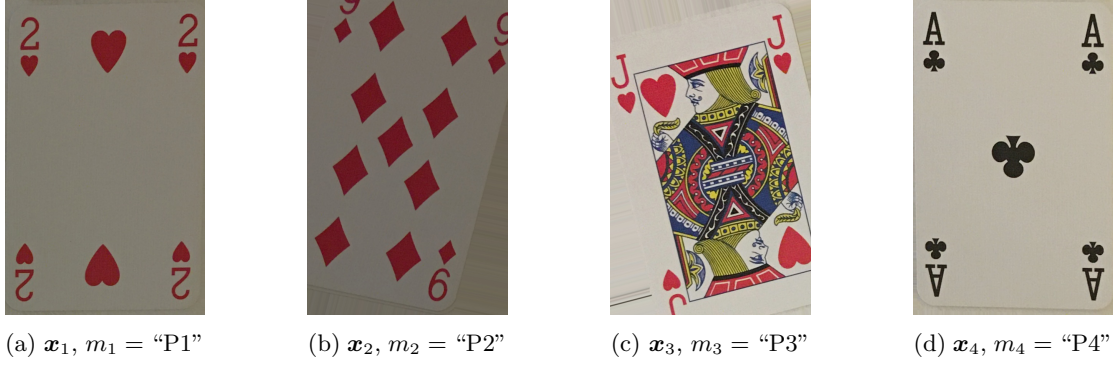


Figure 3.1: Example inputs $\mathbf{x}_i$, and associated meta-data m_i for the Follow Suit Winner task. The meta-data indicates which player played each card. In this example, the label is $y = 3$.

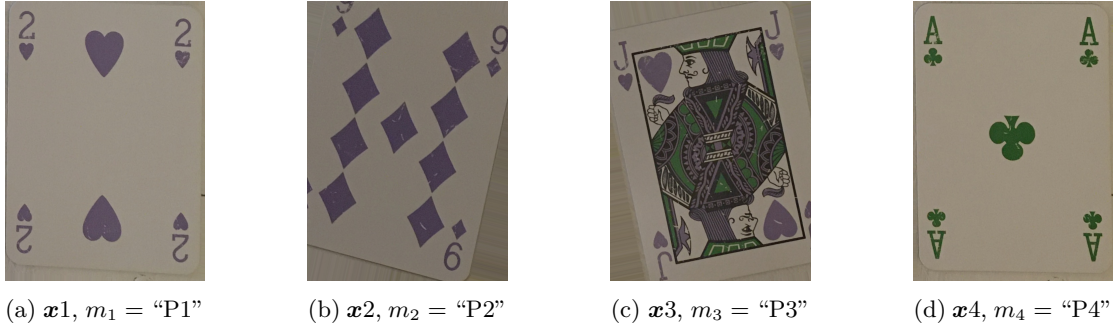


Figure 3.2: Example $\mathbf{x}_i$ and m_i inputs for the Follow Suit Winner task with a distributional shift applied. In this case, standard card images are substituted with images from an alternative deck.

the correct H is learned, the reasoning function h maps a sequence of pairs of predictions and meta-data, i.e., $\langle (g(\mathbf{x}_1), m_1), \dots, (g(\mathbf{x}_4), m_4) \rangle$, to the winner of the game, which in this case is $y = 3$ (Player 3). In this task, g can be pre-trained using standard card images, such as those shown in Figure 3.1, alongside ground truth concept labels (i.e., the suit and rank values $\mathcal{Z} = \{1 \dots 52\}$). A distributional shift can then be applied, by substituting standard cards with the corresponding cards from an alternative deck. For example, Figure 3.2 shows the same data point as Figure 3.1, but with a distributional shift applied. This therefore presents a challenge for learning H , as the predictions from g may be incorrect.

Let us now introduce the FF-NSL framework, to explain how g and H are learned.

3.2 FF-NSL Framework

To learn symbolic knowledge, FF-NSL consists of three major components: (1) A **neural network**, which extracts primitive concepts from raw image data. (2) A **Data-to-Knowledge (D2K) generator**, which transforms the neural network output into symbolic training examples. (3) A **symbolic learner**, which learns a set of logical rules, called a *hypothesis*, to solve the desired task. Once trained, these components enable a perception and reasoning architecture to be constructed for inference. An overview of the FF-NSL learning framework, is presented in Figure 3.3. Let us now define each of these components.

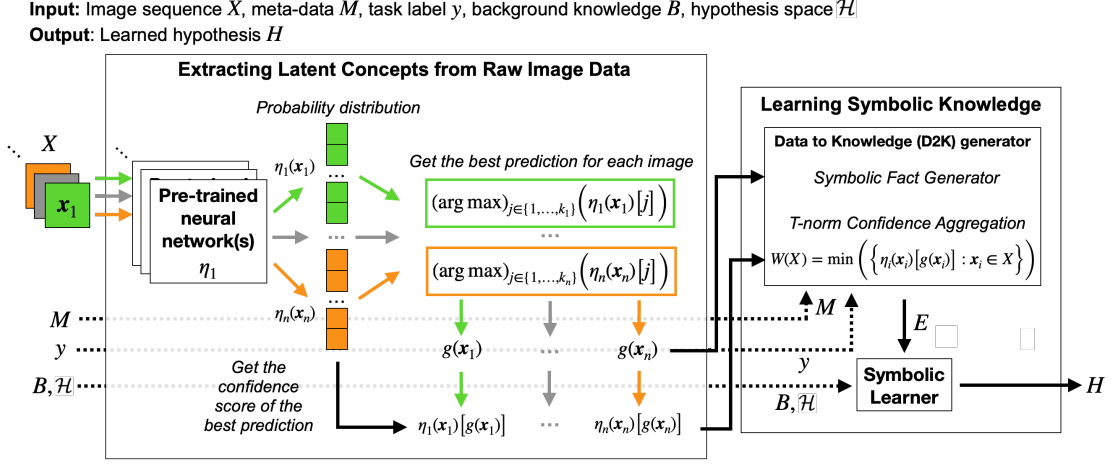


Figure 3.3: General FF-NSL learning framework, shown for a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$.

3.2.1 Extracting Primitive Concepts from Raw Image Data

The perception function g contains a sequence of neural networks $\langle \eta_1, \dots, \eta_n \rangle$, where each η_i extracts a primitive concept from an image $x_i \in X$. Recall that X is the sequence of images observed in a training data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. Depending on the given task, there could be many configurations of the sequence of neural networks. For example, the same neural network may be able to extract the same concept from all images in X , or, each image in X may require a different neural network to extract a different concept. There could also be many combinations thereof, where the same neural network applies to a subsequence of X . Without loss of generality, we denote η_i as the neural network capable of extracting a primitive concept from the image $x_i \in X$. Also, to simplify notation, we assume the neural network outputs a prediction for a single concept. However, depending on the task, a single concept could represent multiple sub-concepts. For example, in the Follow Suit Winner task presented in Example 3.1.1, we allow the neural network to output a prediction that refers to both the suit, and the rank of the playing card. Formally, let us denote a neural network as the function $\eta : \mathcal{X} \rightarrow [0, 1]^k$ which maps a raw image to a k -length vector denoting a probability distribution over k concept values. We define the neural network prediction for an image $x \in X$, as the index in the output vector with the largest probability:

$$g(x) = \arg \max_{j \in \{1, \dots, k\}} (\eta(x)[j]) \quad (3.2)$$

This forms the output of the perception function g . We also require the associated *confidence* of the neural network prediction, which is defined as the maximum value in the probability distribution, or equivalently, the element in the probability distribution at the index $g(x)$:

$$\max(\eta(x)) = \eta(x)[g(x)] \quad (3.3)$$

FF-NSL assumes η is pre-trained. This is achieved using a dataset $\mathcal{D}_{\text{image}} = \{\langle x, z \rangle, \dots\}$, where $x \in \mathcal{X}$ is a raw image, and $z \in \mathcal{Z}$ is a primitive concept label. A loss function is used to compare the prediction for a given image, to the ground-truth concept label. Gradients are then calculated based on the loss, and the parameters of the neural network can be optimised via back-propagation. We denote the parameters as θ , and the parameterised neural network as η^θ . The training objective can therefore be expressed as:

$$\theta^* = \arg \min_{\theta} \left(\sum_{\langle \mathbf{x}_i, z_i \rangle \in \mathcal{D}_{\text{image}}} \text{loss}(\eta_i^{\theta}(\mathbf{x}_i), z_i) \right) \quad (3.4)$$

where “loss” is a desired loss function (e.g., cross-entropy). Intuitively, this objective finds the set of neural network parameters that minimises the prediction error, such that the ground truth concept labels are predicted correctly for each image. For a detailed overview of neural network training, see Chapter 2.

3.2.2 Learning Symbolic Knowledge

To learn symbolic knowledge, a set of training examples E are required for the symbolic learner. A given background knowledge B , and a search space $\mathcal{H}$ are also required, which respectively contain any optional relations relevant for the task, and an enumeration of relevant rules in the particular task domain. In the context of FF-NSL, a training example contains three major components: (1) A set of symbolic facts which represent the neural network predictions, alongside the associated meta-data. (2) An aggregated confidence score which represents a level of confidence of the network predictions (3) The downstream task label, which is given in the data. These three components are transformed into a training example using a D2K generator. The transformation of the neural network predictions into symbolic facts is task specific, and also depends on the particular representation used by the chosen symbolic learner. This is also the case for the encoding of the downstream task label. In general, given a data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$, the D2K generator outputs a training example $e \in E$:

$$e = \text{D2K}(\{\langle g_i(\mathbf{x}_i), \eta_i(\mathbf{x}_i), m_i \rangle : \mathbf{x}_i \in X, m_i \in M\}, y) \quad (3.5)$$

The D2K generator includes a fact generator, and a confidence aggregator. We now define a general aggregation function, which is used by the D2K generator internally, for calculating the aggregated confidence of a training example. This is defined for an image sequence X as:

$$W(X) = \min(\{\eta_i(\mathbf{x}_i)[g(\mathbf{x}_i)] : \mathbf{x}_i \in X\}) \quad (3.6)$$

which is a generalisation of the binary Gödel t -norm used in fuzzy logic to encode fuzzy conjunctions (Metcalf et al., 2008). The Gödel t -norm computes the minimum value from the set of confidence scores that correspond to the neural network prediction for each image in the input sequence. We rely upon the associativity property of t -norms to recursively compute an aggregated value, given $n > 2$ network predictions (see Chapter 2). The Gödel t -norm is chosen over the Product and Łukasiewicz t -norms because it is easy to interpret, and it does not tend towards 0 as n increases, which is the case with the Product t -norm. Although one could adopt a more complicated approach to handle the product case, such as applying a log transformation, we opt for the more interpretable Gödel t -norm. Finally, we find each conjunctive t -norm results in similar performance in our experiments.

The generality of FF-NSL allows it to be instantiated using different neural and symbolic components, depending on the given task. The requirements are, the symbolic learner must support contextual facts within its training examples, and must support learning from noisy, and weighted examples. The neural network is required to output a probability distribution over primitive concept values. As discussed, we consider the LAS symbolic learning systems, since they have achieved state-of-the-art performance when learning from noisy examples (Law et al., 2018), and also support different types of symbolic learning, namely observational, and non-

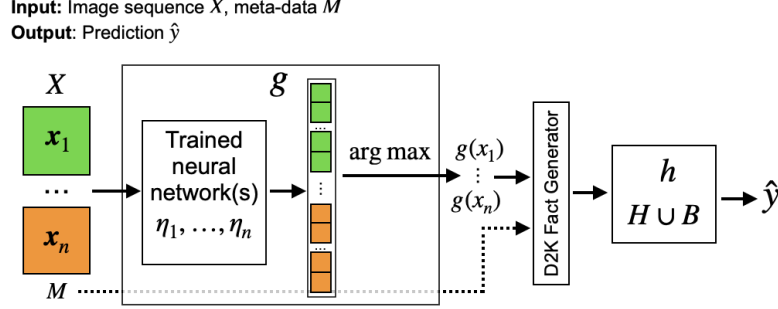
observational predicate learning (see Section 3.3 for example tasks that highlight this difference). Let us now instantiate the D2K generator with respect to the LAS systems, to enable the LAS systems to be used within the FF-NSL framework. We refer to this instantiation as $D2K_{LAS}$. $D2K_{LAS}$ generates a training example e in the form of a WCDPI (Law et al., 2018). The neural network predictions, and the associated meta-data, define the context of a WCDPI. The aggregated confidence score defines the WCDPI weight, and the downstream label defines the inclusion and exclusion sets of the WCDPI, or whether the example is positive or negative, in the case of a binary classification task. Formally, given a training data point $\langle X, M, y \rangle_j \in \mathcal{D}_{task}$, and the background knowledge B , the components of a WCDPI $e = \langle e_{id}, e_{pen}, e_{pi}, e_{ctx} \rangle$ are defined as follows:

- e_{id} : The example identifier, which is uniquely associated to a single data point in the training set.
- e_{pen} : The example weight, which is defined using a transformation on the aggregated confidence score: $e_{pen}(X) = \lfloor 100 \times W(X) \rfloor + 1$, and is required because the LAS systems require integer weight values > 0 . This contributes to the score of a hypothesis, for each example not covered by the hypothesis. In the context of FF-NSL, a high example weight reflects a high aggregated confidence score from the neural network, and therefore, the symbolic learner is inclined to cover symbolic training examples with concepts predicted with high confidence.
- e_{pi} : The partial interpretation $\langle e_{inc}, e_{exc} \rangle$, which contains a set of inclusion and exclusion atoms respectively, in ASP form. These are defined using the label y for multi-class classification tasks. For binary classification tasks, e_{inc} and e_{exc} remain empty. For example, in a simple multi-class classification task that aims to classify an animal as either a horse, dog, or cat, if the label is a horse, $e_{pi} = \langle \{\text{horse.}\}, \{\text{dog.}, \text{cat.}\} \rangle$
- e_{ctx} : The context, which is a set of atoms in ASP form, defined using the concept predictions from the perception function, and the associated meta-data.

The third component of FF-NSL is the symbolic learner itself. The input is a set of training examples E constructed from the output of the D2K generator for each input data point, and a set of background knowledge B , which includes any optional relations relevant to the task. A hypothesis search space $\mathcal{H}$ is also required. The symbolic learner searches for an optimal inductive hypothesis $H \subset \mathcal{H}$ that is consistent with the training examples, and any background relations provided. In the case of the LAS symbolic learners, H is learned according to the optimisation function in Equation 2.27. Finally, once H is learned, H forms part of the reasoning function $h = H \cup B$.

3.2.3 Inference

Once g and H are learned, inference can be performed for new, unseen data points $\langle X, M \rangle$, to predict the output $\hat{y}$, i.e., $\hat{y} = h(\langle (g(\mathbf{x}_1), m_1), \dots, (g(\mathbf{x}_n), m_n) \rangle)$. h is implemented using a logic program, and in our case, we encode the learned H , alongside the background knowledge B into an ASP program. Note that when FF-NSL is instantiated with a particular symbolic learner, H and B are of a certain symbolic representation. Therefore, during inference, the neural network predictions and associated meta-data are converted into a suitable form, using the fact generator within the D2K. An overview of the FF-NSL architecture during inference is presented in Figure 3.4.

Figure 3.4: Inference for a single data point $\langle X, M \rangle$.

3.3 Neuro-Symbolic Evaluation Tasks

In our evaluation, we consider four different tasks, called *Follow Suit Winner*, *Sudoku Grid Validity*, *Crop Yield Prediction* and *Indoor Scene Classification* respectively. The Follow Suit Winner task requires *predicate invention*, as a predicate required in the learned hypothesis is not directly observed in the examples, but is linked to the examples through the background knowledge. This task also requires learning negation as failure. We choose to evaluate a task with these properties to demonstrate the expressiveness of the LAS symbolic learners, since other symbolic learning systems struggle to learn such programs (Cropper and Dumančić, 2022). The remaining tasks require learning in larger hypothesis spaces, which is challenging given the high likelihood for noisy input. The Follow Suit Winner task contains 96 rules in the search space (and therefore 2^{96} hypotheses), whereas the Sudoku Grid Validity, Crop Yield Prediction, and Indoor Scene Classification tasks contain 4451, 194,531 rules, and 809 rules respectively. The FastLAS system has been shown to scale well to very large hypothesis spaces (Law et al., 2020a), and we consider these tasks in order to demonstrate its capability. The Indoor Scene Classification task requires a more advanced perception component to extract a concept from real-world imagery. This task is chosen to demonstrate the applicability of FF-NSL to real-world datasets, and also the generality of FF-NSL, in terms of the range of perception components it can support. Due to the different types of symbolic learning, and the different properties of each task, we instantiate FF-NSL with the different LAS systems, ILASP and FastLAS. We also select these tasks as generally speaking, existing tasks within the neuro-symbolic literature are either restricted to smaller search spaces, or only require learning definite programs.

In this section, we introduce each of these tasks, as well as analyse the accuracy and confidence score distribution of the neural network predictions for the images in the data points used to learn a hypothesis, i.e., the images in the sequence X , for each $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. Since the networks are pre-trained using $\mathcal{D}_{\text{image}}$, which contains different images than those in $\mathcal{D}_{\text{task}}$, we can investigate the reliability of each network when extracting concepts from unseen input data outside the training distribution. This outlines the challenge faced by the symbolic learner, when learning symbolic knowledge. We pre-train two networks: (1) A standard Softmax based CNN, which is widely used to perform image classification, and, (2) EDL-GEN (Sensoy et al., 2020), a state-of-the-art neural network based upon evidential deep learning, which is capable of better representing uncertainty in the neural network predictions. For the Indoor Scene Classification task, we only pre-train one network, Semantic Aware Scene Recognition (SASR) (López-Cifuentes et al., 2020), which is a more advanced dual-branch CNN specific to the task, as it is trained to output scene level classifications, utilising semantic segmentation information, and raw RGB data on each CNN branch respectively. Let us now describe the tasks

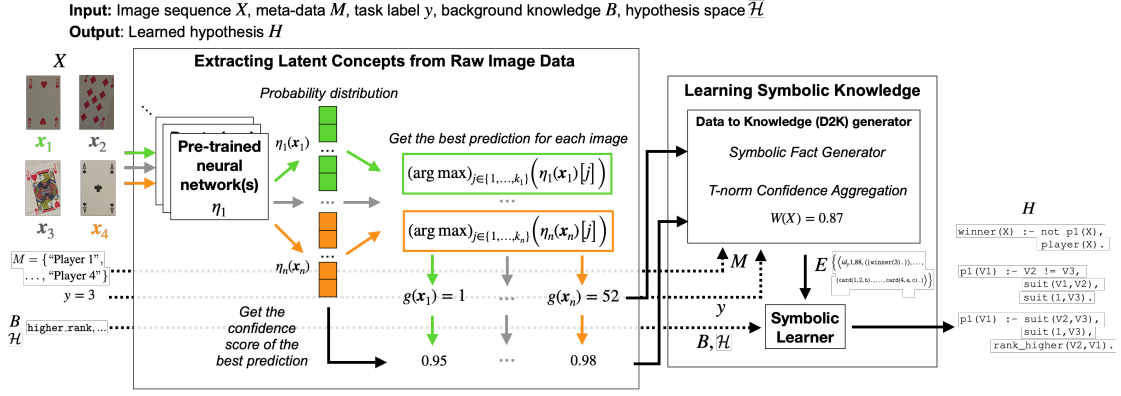


Figure 3.5: FF-NSL learning framework, instantiated for a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ in the Follow Suit Winner task.

we consider in our evaluation, and present the neural network performance under distributional shifts, for each of the four tasks.

Follow Suit Winner

Following the definition of the Follow Suit Winner task in Example 3.1.1, let us now present an example WCDPI representation output by D2K_{LAS} . An instantiated version of the FF-NSL framework for this task, that corresponds to the example described here, is shown in Figure 3.5. We assume the card images shown in Figure 3.1, and the perception function predicts the correct cards $g(x_1) = 1$, $g(x_2) = 21$, $g(x_3) = 10$, and $g(x_4) = 52$, representing the “Two of Hearts”, “Nine of Diamonds”, “Jack of Hearts”, and the “Ace of Clubs” respectively. The meta-data M indicates which player played each card. To compute the aggregated confidence score, let us assume the neural network predicts with the following confidence scores for each image: $\eta_1(x_1)[g(x_1)] = 0.95$, $\eta_2(x_2)[g(x_2)] = 0.87$, $\eta_3(x_3)[g(x_3)] = 0.92$, and $\eta_4(x_4)[g(x_4)] = 0.98$. The aggregated confidence score is the minimum predicted score, i.e., $W(X) = 0.87$. The WCDPI example weight penalty is calculated by D2K_{LAS} as $e_{\text{pen}} = \lfloor 100 \times 0.87 \rfloor + 1 = 88$. D2K_{LAS} also encodes the downstream label as the partial interpretation $e_{\text{pi}} = \langle \{\text{winner}(y)\}, \{\text{winner}(y') : y' \in \mathcal{Y}, y' \neq y\} \rangle$. For our example, $e_{\text{pi}} = \langle \{\text{winner}(3)\}, \{\text{winner}(1), \text{winner}(2), \text{winner}(4)\} \rangle$, which denotes the only winner is Player 3. In this task, all LAS training examples are positive, and the label is encoded using the inclusion and exclusion sets. Finally, the concepts and meta-data are encoded as contextual atoms in e_{ctx} . For our example, $e_{\text{ctx}} = \{\text{card}(1, 2, \text{h}), \text{card}(2, 9, \text{d}), \text{card}(3, \text{j}, \text{h}), \text{card}(4, \text{a}, \text{c})\}$, representing the rank and suit values of the card played by each player. Therefore, the complete WCDPI training example e for our $\langle X, M, y \rangle_j \in \mathcal{D}_{\text{task}}$ data point output by D2K_{LAS} is given as:

$$e = \langle \text{"id_j"}, 1, 88, \langle \{\text{winner}(3)\}, \{\text{winner}(1), \text{winner}(2), \text{winner}(4)\} \rangle, \langle \text{card}(1, 2, \text{h}), \text{card}(2, 9, \text{d}), \text{card}(3, \text{j}, \text{h}), \text{card}(4, \text{a}, \text{c}) \rangle \rangle \quad (3.7)$$

In this task, we use the LAS symbolic learner ILASP (Law, 2018; Law et al., 2020b), since ILASP supports learning predicate invention. The full ASP specification of the background knowledge, and the hypothesis search space for this task is presented in Listing 3.1. B encodes possible suit and rank values and the four players (line 2), a `higher_rank` predicate that defines when a player has a higher ranked card than another player (line 5), and a link predicate to

match the suit of a given player to the corresponding `card` predicate given in e_{ctx} (line 8). The hypothesis search space $\mathcal{H}$ (lines 11-25) includes the following predicates: `player`, `suit`, `higher_rank`, `winner`, to denote the winner of the game, and the ‘!=’ relation to denote two suit values are not equal. We also include an instruction to perform predicate invention (lines 15–17), which enables the `winner` of the game to be defined in terms of the `card` predicates specified in the examples.

Listing 3.1: Follow Suit Winner Background Knowledge and Hypothesis Search Space Specification.

```

1 % Background knowledge
2 suit(h). suit(s). suit(d). suit(c). rank(2..10). rank(j). rank(q). rank(k). rank(a). rank_value(2..10, 2..10).
   rank_value(j, 11). rank_value(q, 12). rank_value(k, 13). rank_value(a, 14). player(1..4).
3
4 % Higher rank rule
5 rank_higher(P1, P2) :- card(P1, R1, _), card(P2, R2, _), rank_value(R1, V1), rank_value(R2, V2), V1 > V2.
6
7 % Link player's card to suit
8 suit(P1, S) :- card(P1, _, S).
9
10 % Mode declarations to specify hypothesis search space
11 P(X) :- Q(X), identity(P, Q).
12 P(X) :- player(X), not Q(X), inverse(P, Q).
13 #modeb(2, inverse(target/1, invented/1)).
14 #modeb(2, identity(target/1, invented/1)).
15 #predicate(target, winner/1).
16 #predicate(invented, p1/1).
17 #modeb(p1(var(player))).
18 #modeb(1, var(suit) != var(suit)).
19 #modeb(1, suit(var(player), var(suit)), (positive)).
20 #modeb(1, suit(const(player), var(suit)), (positive)).
21 #modeb(1, rank_higher(var(player), var(player)), (positive)).
22 #constant(player, 1).
23 #constant(player, 2).
24 #constant(player, 3).
25 #constant(player, 4).

```

As for the neural network performance, we start by pre-training the Softmax and EDL-GEN neural networks on standard playing card images, and apply *minor* and *major* distributional shifts by substituting standard playing card images with images from alternative decks. Example images are shown in Figure 3.6 for the queen of hearts card, taken from the *Standard* (3.6a), *Batman Joker* (3.6b), *Captain America* (3.6c), *Adversarial Standard* (3.6d), *Adversarial Batman Joker* (3.6e) and *Adversarial Captain America* (3.6f) decks. The Batman Joker and Captain America decks represent *minor* distributional shifts, and the adversarial decks represent *major* distributional shifts, where card images from each of the Standard, Batman Joker and Captain America decks are placed against a background containing additional card images from the Standard deck. These adversarial decks are designed to trick the neural networks into predicting incorrectly, as images from the Standard deck are from the same distribution as the card images used during pre-training. In order to understand the level of noise the symbolic learner has to tolerate, Figure 3.7 presents the accuracy and confidence score distribution of both Softmax and EDL-GEN neural networks, when evaluated on a test set of different playing card decks than the Standard deck used for pre-training.

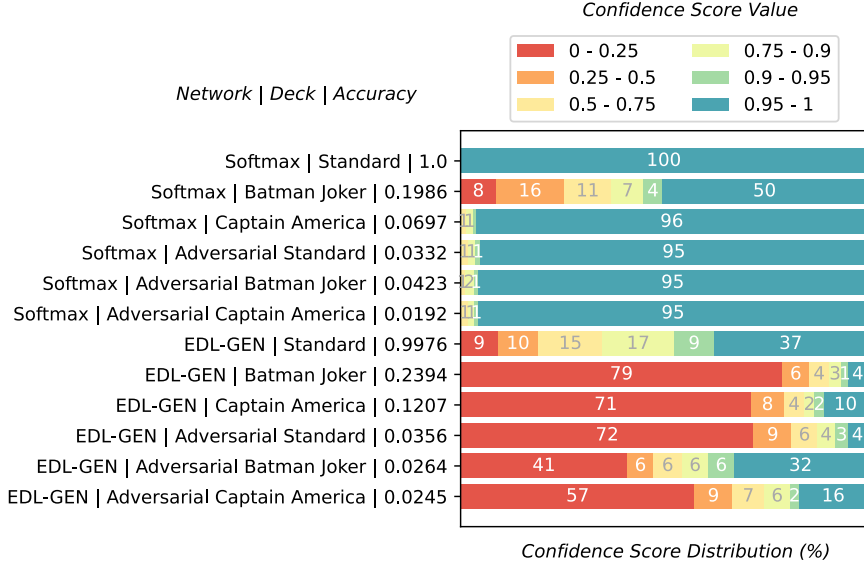


Figure 3.7: Neural network performance under distributional shifts in the Follow Suit Winner task. Each row presents the accuracy of a neural network on a particular card deck. The confidence score distribution reflects the percentage of neural network predictions within the varying confidence intervals. For example, in the Softmax Captain America case, 96% of the neural network predictions were made with confidence in the interval $[0.95, 1]$, despite the network only achieving 0.0697 accuracy.

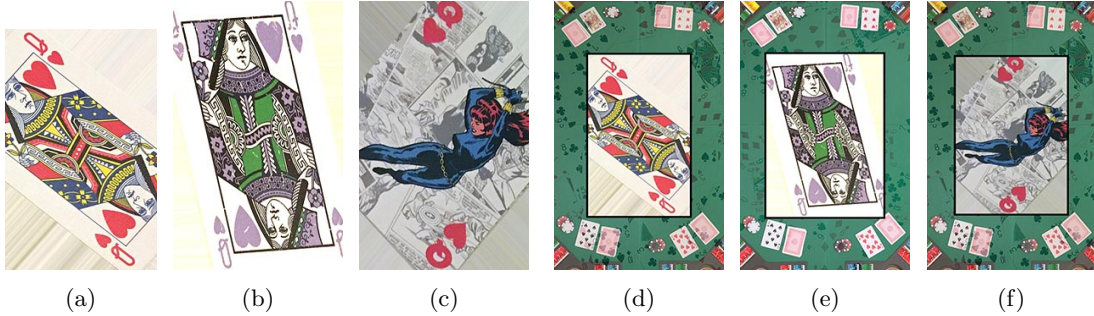


Figure 3.6: Example playing card images used in the Follow Suit Winner task.

Each row shows the type of neural network, the playing card deck used for evaluation, the predictive accuracy, and the confidence score distribution. As one would expect, the accuracy is very high when classifying playing card images from the standard deck, as this is the deck used for pre-training. For the Softmax neural network, the confidence score is also very high in this case, whereas EDL-GEN has more distributed confidence scores, which demonstrates the EDL-GEN neural network expresses some uncertainty over its predictions (it is also slightly less accurate than Softmax). When evaluating the pre-trained networks on the other decks, the Softmax neural network still reports high confidence, despite its overall low accuracy. The EDL-GEN network on the other hand, has comparable low accuracy but with much lower confidence. For example, evaluating on the *Captain America* deck (see 3rd and 9th rows), 96% of Softmax predictions are made with confidence in the interval $[0.95, 1]$, despite an accuracy of 0.0697, whereas only 10% of EDL-GEN predictions are made within this same confidence interval. The accuracy of the EDL-GEN network is 0.1207, i.e., $\sim 12\%$ of the predictions are correct. Inspecting the confidence score distribution, 12% of the predictions are made with confidence ≥ 0.9 . This indicates a significant improvement in confidence estimates, given the low performance of the

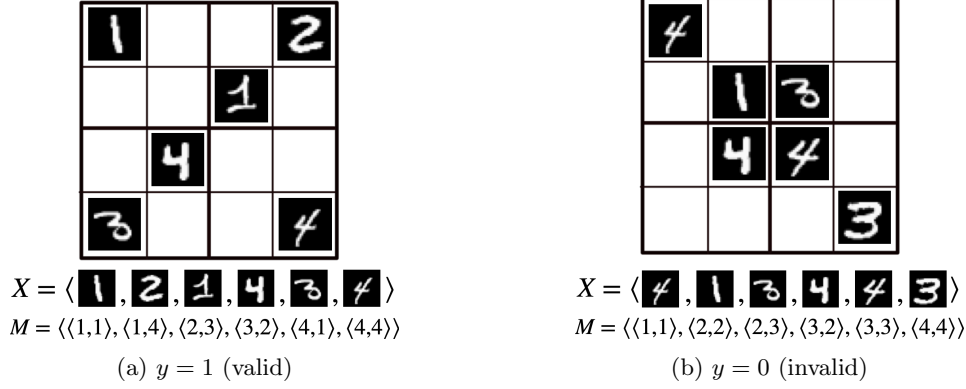


Figure 3.8: Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Sudoku Grid Validity 4×4 task.

network. As for the overall accuracy, EDL-GEN performs slightly better than Softmax, given decks representing minor distributional shifts, whereas both networks perform similarly, when applied to decks representing major distributional shifts. This highlights the challenge for the symbolic learner, as the neural network predictions are likely to be incorrect, and in the case of a Softmax-based neural network, may potentially be made with high confidence.

Sudoku Grid Validity

Our second task is the Sudoku Grid Validity task, where the goal is to learn a hypothesis that can predict whether a given Sudoku grid is valid or not. A grid is considered valid if there are no repeating digits in any row, column, or block. A training example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ consists of a sequence of images X of handwritten MNIST digits (Deng, 2012), corresponding to the digits in a Sudoku grid. We assume the original Sudoku grid has been pre-processed to return images of the digits in different cells. The meta-data M defines the row and column coordinates of each digit image, and the downstream label y is binary, indicating whether the grid is valid ($y = 1$), or invalid ($y = 0$). We consider both 4×4 and 9×9 Sudoku grids as separate sub-tasks, although not all of the cells in the Sudoku grid contain a digit, some remain blank. Example data points for the 4×4 task are shown in Figure 3.8.

In this task, we pre-train two neural networks, one for the 4×4 task, to predict MNIST digits 1–4, and one for the 9×9 task, to predict MNIST digits 1–9. This is to evaluate a simpler perception task with 4×4 grids, since there are fewer digit classes for the network to predict, as well as a more challenging perception task with 9×9 grids. Therefore, the neural network outputs a probability distribution $[0, 1]^4$, and $[0, 1]^9$ for the 4×4 , and 9×9 tasks respectively. The neural networks are pre-trained using standard MNIST images, and the same neural network can be used to extract the digit concept from each image in a grid. Considering Figure 3.8a, and, assuming the perception function predicts correctly, $g(\mathbf{x}_1) = 1$, $g(\mathbf{x}_2) = 2$, $g(\mathbf{x}_3) = 1$, $g(\mathbf{x}_4) = 4$, $g(\mathbf{x}_5) = 3$, and $g(\mathbf{x}_6) = 4$. Let us also assume the neural network predicts with the following confidence scores: $\eta_1(\mathbf{x}_1)[g(\mathbf{x}_1)] = 0.91$, $\eta_2(\mathbf{x}_2)[g(\mathbf{x}_2)] = 0.93$, $\eta_3(\mathbf{x}_3)[g(\mathbf{x}_3)] = 0.85$, $\eta_4(\mathbf{x}_4)[g(\mathbf{x}_4)] = 0.75$, $\eta_5(\mathbf{x}_5)[g(\mathbf{x}_5)] = 0.98$, and $\eta_6(\mathbf{x}_6)[g(\mathbf{x}_6)] = 0.96$. The aggregated confidence score $W(X) = 0.75$. The meta-data M indicates the row and column of each digit image.

Considering the $\langle X, M, y \rangle$ data point shown in Figure 3.8a, a WCDPI example e output by D2K_{LAS} is defined as follows. $e_{\text{pen}} = \lfloor 100 \times 0.75 \rfloor + 1 = 76$, and $e_{\text{pi}} = \langle \{\}, \{\} \rangle$, since this is a binary classification task and the examples can be positive or negative (according to the label y). The predicted concepts, and meta-data, are encoded into the example context:

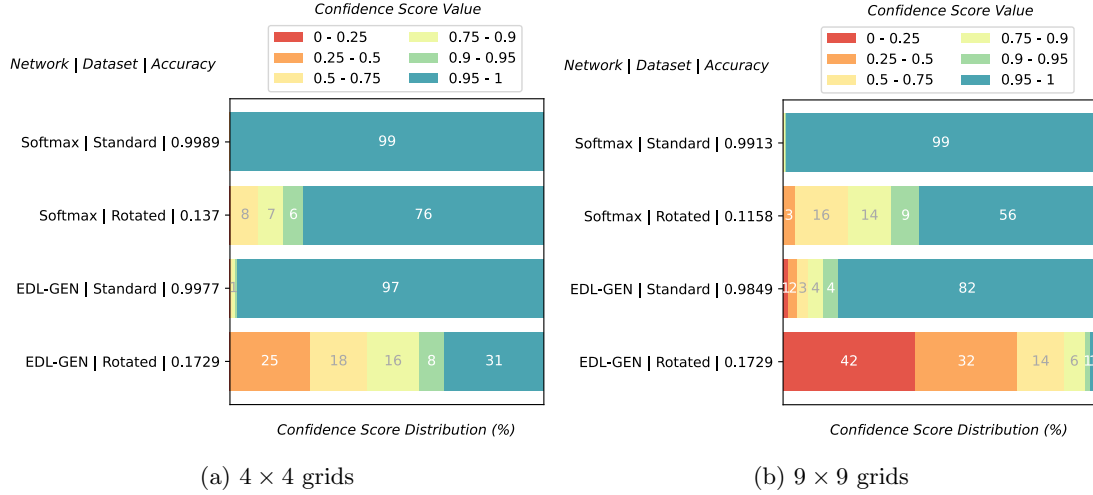


Figure 3.9: Neural network performance under distributional shifts, Sudoku Grid Validity task. Each row presents the accuracy of a neural network with and without the distributional shift (rotation) applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval. For example, in the 4x4 Softmax rotated case, 76% of neural network predictions are made with confidence in the interval [0.95, 1], despite the network only achieving 0.137 accuracy.

$e_{\text{ctx}} = \{\text{value}("1,1",1), \text{value}("1,4",2), \text{value}("2,3",1), \text{value}("3,2",4), \text{value}("4,1",3), \text{value}("4,4",4)\}$, where the first argument " a,b " represents the a -th row and b -th column of the grid, and the second argument represents the digit value.

In this task, we use the FastLAS symbolic learner (Law et al., 2020a), as FastLAS has been shown to be more scalable than ILASP for tasks of this nature (i.e., *observational* predicate learning tasks) (Law et al., 2020a). In both 4x4 and 9x9 tasks, we encode the layout of the grid within the background knowledge B , i.e., the row, column, and block identifiers of each cell, as well as a `neq` relation that states two cell identifiers are not equal. The hypothesis search space $\mathcal{H}$ includes the relations `value` (expressed in e_{ctx}), `neq`, `row`, `column`, and `block`, as well as their negations `not row`, `not column`, and `not block`. The full ASP specification of the background knowledge, and hypothesis search space, are given in Appendix A.

In terms of the neural networks, we first pre-train both Softmax and EDL-GEN neural networks on standard images from the MNIST training set. In all experiments, we use MNIST digits 1-4, and 1-9 for the respective Sudoku grid sizes. Figure 3.9 shows the accuracy and confidence score distribution of the pre-trained neural networks, on two test sets: a *standard* MNIST test set, and a test set where the MNIST digits are *rotated* 90° clockwise, representing a distributional shift. The test sets also contain MNIST digits 1-4 or 1-9, depending on the Sudoku grid size. The results are similar to the Follow Suit Winner task. The Softmax neural network predicts with high confidence given data subject to distributional shift, despite its low test set accuracy. The EDL-GEN neural network predicts more accurately than Softmax on data subject to distributional shift, but Softmax was slightly more accurate on the in-distribution standard test sets.

Crop Yield Prediction

We also evaluate FF-NSL on a task closer to a real-world problem domain. The goal is to predict the quality of a crop’s yield, given a photograph of a crop’s leaf, and also some symbolic information regarding the location of the crop. Each leaf image may represent a healthy or

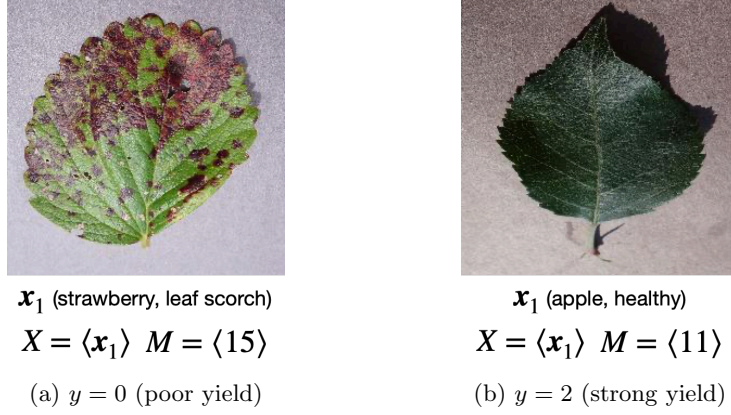


Figure 3.10: Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Crop Yield Prediction task.

diseased crop of a certain species. A training example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ consists of a single element sequence X containing the crop image, a single element sequence of meta-data M encoding the location of the crop, and a downstream label $y \in \{0, 1, 2\}$ indicating whether the yield is *poor* ($y = 0$), *moderate* ($y = 1$), or *strong* ($y = 2$). Example data points are shown in Figure 3.10, for a strawberry crop with the leaf scorch disease (Figure 3.10a), and a healthy apple crop (Figure 3.10b).

In this task, we pre-train a neural network to extract the species and disease concepts directly from a leaf image. There are 38 possible species and disease combinations, and therefore the neural network outputs the probability distribution $[0, 1]^{38}$. As in the Follow Suit Winner task, we maintain a mapping from the neural network prediction to the species and disease names. The mapping is arranged in alphabetical order, where the species names are arranged first, followed by the disease names. For example, prediction 1 corresponds to “*apple, apple scab*”, and prediction 38 corresponds to “*tomato, healthy*”. Considering Figure 3.10a as an example, and assuming the perception function predicts correctly, $g(\mathbf{x}_1) = 28$. Let us also assume the neural network predicts with confidence $\eta_1(\mathbf{x}_1)[g(\mathbf{x}_1)] = 0.89$. The aggregated confidence score $W(X)$ is also 0.89, since X only contains a single image. The meta-data M indicates the location ID of the particular crop, in this case, let us assume this is 15. For this example, D2K_{LAS} outputs a WCDPI example defined as follows. $e_{\text{pen}} = \lfloor 100 \times 0.89 \rfloor + 1 = 90$, $e_{\text{pi}} = \langle \{\text{yield}(0).\}, \{\text{yield}(1)., \text{yield}(2).\} \rangle$, and $e_{\text{ctx}} = \{\text{location}(15)., \text{species}(\text{strawberry})., \text{disease}(\text{leaf_scorch}).\}$. In this task, all training examples are positive, and the FastLAS symbolic learner is used, since it is able to learn a hypothesis quicker than ILASP on this task. The background knowledge B contains a constraint to ensure the learned hypothesis performs a classification, i.e., it outputs exactly one crop yield value given a crop location, species and disease. The hypothesis search space $\mathcal{H}$ includes the `location`, `species`, and `disease` relations. The full ASP specification is given in Appendix A.

For imagery, we use the Plant Village dataset containing images of healthy and diseased crops (Hughes and Salathé, 2015). A distributional shift is applied to crop images using a hue filter, after the softmax and EDL-GEN neural networks are pre-trained on the unaltered plant images. The results are shown in Figure 3.11, where the Softmax neural network achieves 0.88 accuracy for the standard images, and performs poorly when classifying images subject to a hue-shift. In both datasets, the neural network predictions are made with very high confidence. The EDL-GEN neural network achieves 0.831 accuracy on the standard images, and 0.318 accuracy on the hue shift images, which is much higher than Softmax. Crucially, the EDL-GEN neural

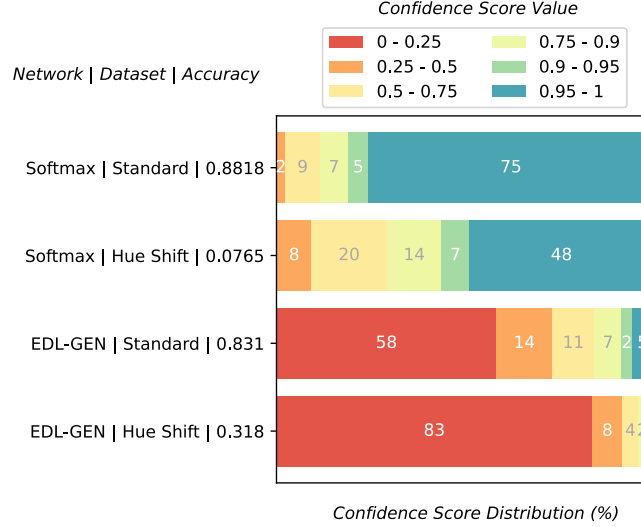


Figure 3.11: Neural network performance under distributional shifts. Crop Yield Prediction task. Each row presents the accuracy of a neural network with and without the distributional shift (hue filter) applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval. For example, in the Softmax hue shift case, 48% of neural network predictions are made with confidence in the interval $[0.95, 1]$, despite the network only achieving 0.0765 accuracy.

network predicts with much lower confidence than Softmax on both datasets which better reflects the predictive uncertainty. However, the confidence for standard images was somewhat lower than expected, as 58% of predictions are made with less than 0.25 confidence, despite 0.831 accuracy.

Indoor Scene Classification

Our final task evaluates FF-NSL with real-world data, and a more challenging perception requirement. The goal is to learn a hypothesis that maps scene-level classifications into higher-level super-classes that correspond to a collection of scenes, given images of indoor scenes. For example, images of a bedroom, bathroom, or living room are considered scene-level, and the super-class is “home”. A training example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ consists of a single element sequence X containing the indoor scene image, an empty sequence of meta-data M , and a downstream label $y \in \{0, \dots, 4\}$ to represent the super-class “store”, “home”, “public space”, “leisure”, and “working place” respectively. Example data points are shown in Figure 3.12.

In this task, we pre-train a single neural network to extract the scene-level concept from an image. There are 67 possible scenes in total, and therefore, the neural network outputs the probability distribution $[0, 1]^{67}$. We maintain a mapping from the neural network prediction to the type of indoor scene. The mapping is arranged in alphabetical order. For example, prediction 1 corresponds to “airport inside”, and prediction 67 corresponds to “wine cellar”. Consider Figure 3.12a as an example, and assume the perception function predicts correctly, $g(\mathbf{x}_1) = 8$. Let us also assume the neural network predicts with confidence $\eta_1(\mathbf{x}_1)[g(\mathbf{x}_1)] = 0.77$. The aggregated confidence score $W(X)$ is also 0.77, since X only contains a single image. In this task, the meta-data M is empty. In this example, D2K_{LAS} outputs a WCDPI defined as follows. $e_{\text{pen}} = \lfloor 100 \times 0.77 \rfloor + 1 = 78$, $e_{\text{pi}} = \{\{\text{label}(3).\}, \{\text{label}(0).\}, \{\text{label}(1).\}, \{\text{label}(2).\}, \{\text{label}(4).\}\}$, and $e_{\text{ctx}} = \{\text{image}(\text{bowling}).\}$. All training examples are positive, and the FastLAS symbolic learner is also used in this task. No background knowledge is required, and the hypothesis search



(a) Bowling scene as image x_1 , $M = \langle \rangle$, $y = 3$ (leisure)



(b) Subway scene as image x_1 , $M = \langle \rangle$, $y = 2$ (public space)

Figure 3.12: Example $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data points for the Indoor Scene Classification task.

space $\mathcal{H}$ includes the **image** relation specified in the example context. The ASP specification of the background knowledge and search space is given in Appendix A.

In this task, the imagery is from the MIT Indoor Scenes dataset (Quattoni and Torralba, 2009). To apply a distributional shift, we transform each image using a Gaussian blur, hue shift, and 180° rotation, after the network is pre-trained on unaltered images. Recall in this task we only use one network, a SASR network (López-Cifuentes et al., 2020), which is a more advanced dual-branch CNN specific to this task, as it is trained to output scene level classifications, utilising semantic segmentation information, and raw RGB data on each CNN branch respectively. The results are shown in Figure 3.13. The distributional shift reduces the network accuracy from 0.87 to 0.11, although the confidence scores from the network appropriately reflect the reduced accuracy, when the distributional shift is applied. Although the SASR network does not have an uncertainty-aware architecture like the EDL-GEN network, it is able to predict with low confidence under our distributional shift. We suspect this is due to the shifted data points falling between the decision boundary of the 67 scene classes, rather than being completely out-of-distribution, enabling the network to better reflect its uncertainty amongst the possible classes.

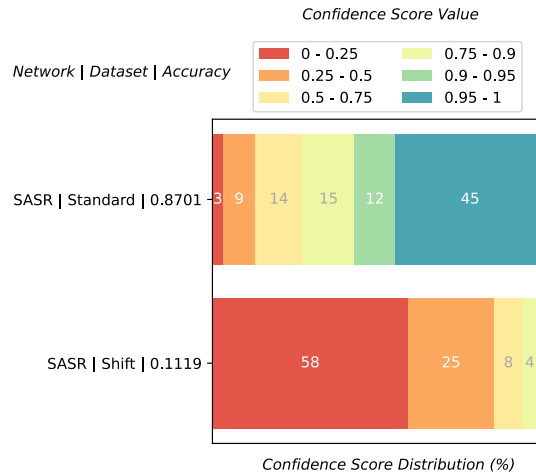


Figure 3.13: Neural network performance under distributional shifts. Indoor Scene Classification task. Each row represents the accuracy of a neural network with and without the distributional shift applied. The confidence score distribution indicates the percentage of neural network predictions with confidence in a certain interval.

3.4 RQ1: Evaluating the Learned Hypotheses

In the following sections, we present our evaluation of FF-NSL when applied to the four problem domains introduced in this section. Our evaluation addresses the following research questions. (1) Can the LAS symbolic learners used within FF-NSL learn accurate hypotheses, even when large percentages of training examples are subject to distributional shifts? This investigates the potential for our framework to *learn* in real-world environments. (2) How does the entire FF-NSL framework perform, when images subject to distributional shifts are observed during inference? This investigates the potential for our framework to perform *inference* in real-world environments. (3) Can FF-NSL be configured to output an estimate of uncertainty during inference, such that a level of confidence can be associated with the downstream predictions? These three questions are answered in the following three sections. In Section 3.4, we evaluate the learned hypothesis accuracy, and perform an in-depth analysis of which aspects of the FF-NSL framework contribute to its performance. In Section 3.5, we evaluate FF-NSL when distributional shifts are observed during inference. Finally, in Section 3.6, we extend the inference capability of FF-NSL with the probabilistic reasoning framework ProbLog (De Raedt et al., 2007), to investigate whether FF-NSL can output robust uncertainty information for its downstream predictions.

3.4.1 Methodology

In this section, we evaluate the accuracy of the learned hypotheses, to investigate whether the symbolic learner is able to tolerate noisy neural network predictions, especially when the raw input is subject to distributional shifts. In each task, we evaluate the learned hypothesis accuracy, interpretability and learning time. The accuracy is measured using a symbolic test set, which contains ground-truth concept values. Given the ground-truth concepts, we compare a downstream prediction output from the hypothesis to the downstream label. This effectively assumes the neural network is 100% accurate, and therefore targets the evaluation towards the hypothesis. As FF-NSL relies upon *pre-trained* neural network(s), we consider the hypothesis accuracy at increasing percentages of $\langle X, M, y \rangle$ training data points subject to distributional shifts, to investigate the robustness of the symbolic learner. Note that the symbolic test set remains unchanged, and is not affected in these cases. In all tasks except Indoor Scene Classification, we use two neural networks, Softmax and EDL-GEN, with the goal of investigating whether an uncertainty-aware network provides any benefit when it comes to learning a hypothesis from (noisy) network predictions. When coupled with the symbolic learner in FF-NSL, we refer to these networks as *FF-NSL Softmax* and *FF-NSL EDL-GEN* respectively. In the Indoor Scene Classification task we use a bespoke network, SASR (López-Cifuentes et al., 2020) due to the advanced perception requirements in this task. Interpretability is measured in terms of the total number of atoms, as a hypothesis with a lower number of atoms is considered to be more interpretable (Lakkaraju et al., 2016). Finally, the learning time is the wall-clock time the symbolic learner requires to learn the hypothesis.

To perform a deeper analysis of the factors that influence the learned hypothesis accuracy, we measure the percentage of WCDPI training examples that contain concepts in the example context e_{ctx} , which are *consistent* with respect to the downstream task label. This enables us to quantify the level of noise given to the symbolic learner, since the neural network(s) may predict incorrectly, but the WCDPI may remain consistent with the target hypothesis. For example, in the Follow Suit Winner task, the neural network may predict the incorrect rank of a certain player’s card, but if the suit is predicted correctly, this may not effect the predicted winner of the

game. In these cases, we say the facts given in the context of the WCDPI (e_{ctx}), are consistent with the downstream label, despite the neural network predictions not being 100% accurate. We also measure the bias given to the symbolic learner in terms of the example weight penalties. To do so, we define the weight penalty ratio r , over the generated examples:

$$r = \frac{\sum_{e \in E_{\text{correct}}} e_{\text{pen}}}{\sum_{e \in E} e_{\text{pen}}} \quad (3.8)$$

where E_{correct} is the set of correctly generated examples, (i.e., those with concepts in e_{ctx} that are consistent with the downstream label), and E is the complete set of generated examples. Ideally, FF-NSL should allocate a higher proportion of the total weight penalty to examples that contain correct neural network predictions. We compare the hypothesis accuracy as learned from examples with neural network confidence based weight penalties, to that when the hypothesis is learned from examples with the same constant weight penalty. This enables us to investigate the effect of the neural network confidence driven weight penalties, compared to a reasonable baseline.

3.4.2 Experimental Setup

Datasets. We generate synthetic datasets for learning a hypothesis, in all tasks except Indoor Scene Classification, which relies upon the real MIT Indoor Scenes dataset (Quattoni and Torralba, 2009). As FF-NSL adopts a sequential pipeline approach for learning g and h , it is important to ensure the datasets used for evaluation are not contaminated with data points already observed during training of one of the components. In all tasks, no images from the training set used to train the neural networks $\mathcal{D}_{\text{image}}$, appear in the set of examples for the symbolic rule learner $\mathcal{D}_{\text{task}}$. Also, all evaluation metrics are reported over a held-out test set, where the test images do not exist in $\mathcal{D}_{\text{image}}$ or $\mathcal{D}_{\text{task}}$. As the LAS symbolic learners are known to be data efficient, meaning they can learn from a small number of examples, whereas the neural and statistical baselines are not, we create small and large training sets, where the large training sets, in the case of Follow Suit Winner and Sudoku Grid Validity, contain $100\times$ more examples than the small datasets. In the Crop Yield Prediction task, the small datasets contain 783 data points, and the large datasets contain 12,717 data points, as we find 12,717 data points are required for the baselines to match the performance of FF-NSL. This highlights the data efficiency gained by our approach. For the Indoor Scene Classification task, only one dataset containing 400 examples is used, since the baselines perform strongly on this dataset. Also, since the training and test split is already defined for this dataset (Quattoni and Torralba, 2009), we evaluate over one repeat. On all other tasks, we generate 5 different training sets, and report the average performance on the held-out test set, given the rules learned from each training set. Given the small dataset sizes, averaging over 5 different training sets ensures any possible bias in training set generation is mitigated.

Hypothesis learning and evaluation. As discussed in Section 3.3, we use the FastLAS symbolic learner for all tasks except Follow Suit Winner, where the ILASP symbolic learner is required to perform predicate invention. Unless otherwise stated, we evaluate the learned hypothesis accuracy over 5 repeats, with 5 different randomly generated training sets. We evaluate the accuracy of the learned hypothesis from each dataset using the same held-out test set, and report the standard error over the 5 repeats.

Baselines. We compare FF-NSL to other hybrid systems. To perform perception, we use the same pre-trained Softmax CNN, as used in FF-NSL Softmax. In the Indoor Scene Classification

task, the same SASR network is used. For rule learning, we evaluate a Random Forest (RF) and a second neural network in all tasks.² These baseline rule learning systems are trained in the same manner as FF-NSL, using the same predictions from the pre-trained perception component, and the downstream task label. The RF is chosen as a powerful decision tree approach, known for being a lightweight model that is quick to train, and also has a certain level of interpretability. In the Follow Suit Winner and Crop Yield Prediction tasks, the neural network rule learning baseline is a Fully Connected Network (FCN), and in the Sudoku Grid Validity task, the neural network is a CNN-LSTM designed for sequence classification problems, where the CNN component can learn spatial dependencies in the Sudoku grid, and the LSTM can predict the grid validity. To measure interpretability of the RF baseline, we examine the first tree in the forest, and extract a rule from each branch from root to leaf. For the neural network baselines, we fitted a surrogate decision tree model to approximate black-box predictions (Molnar, 2019), and applied the same rule extraction method as that used for the RF.

In terms of hyper-parameter tuning, for the RF, we tune the number of estimators across $\{10, 20, 50, 100, 200\}$ for each experiment, by choosing the value that maximises the accuracy of the learned hypothesis on the first training set repeat. The best performing value of 100 is obtained in all experiments.

In the Follow Suit Winner task, the FCN baseline consists of 3 fully connected layers with the ReLU activation function applied to each layer. Dropout is also applied after the first and second layers. Finally, a Softmax layer squashes the final logits into 4 classes, representing each possible winner. To tune the FCN, we sample the number of output units in the first and second layers, i.e., $l1 \in \{20, 32, 46, 52\}$ and $l2 \in \{52, 64, 74, 80\}$ respectively, along with the dropout probability in both dropout layers $dr \in \{0.1, 0.2, 0.5\}$. We sample all possible parameter combinations in a grid-search, and tune on the first *small* dataset, with no data points subject to distributional shift. The best performing parameter values of $l1 = 20$, $l2 = 74$ and $dr = 0.1$ are chosen. These parameters are fixed for all models trained.

In the Sudoku Grid Validity task, the CNN-LSTM takes as input the sequence of predictions from the perception model, encodes them using an embedding layer, and then applies a 1D convolution with a kernel size of 3, and the ReLU activation function. Then, a 1D max pooling layer with pool size 2 is used, followed by a dropout layer, an LSTM layer and a second dropout layer. Finally, a dense fully connected layer with the sigmoid activation function is used to produce a binary classification of the input digit sequence. The input sequence length to the embedding layer is 16 for 4×4 grids and 81 for 9×9 grids, representing each cell on the Sudoku grid. To tune the CNN-LSTM, we sample the learning rate $lr \in \{0.1, 0.001, 0.0001\}$, the embedding dimension of the embedding layer $ed \in \{32, 96, 256\}$, the number of output channels of the 1D convolution layer $oc \in \{64, 96\}$, the number of hidden features in the LSTM layer $lh \in \{32, 96, 128\}$ and the dropout probability $dr \in \{0.01, 0.05, 0.1\}$. We perform 10 samples and evaluate the model on the first *large* dataset with 0 data points subject to distributional shift, training for 2 epochs. The best performing parameter values of $lr = 0.0001$, $ed = 96$, $oc = 64$, $lh = 96$ and $dr = 0.01$ are chosen, and are fixed for all models trained.

In the Crop Yield Prediction, and Indoor Scene Classification tasks, the FCN baseline is the same model architecture as used in the Follow Suit Winner task, and we perform a similar hyper-parameter tuning run. The best performing parameter values of $l1 = 20$, $l2 = 32$ and $dr = 0.1$ are obtained for both tasks, and fixed for all models trained. Let us now describe more details of each problem domain, and present our experimental results, using the evaluation

²Note: This is a fully neural system as there is also a neural perception component.

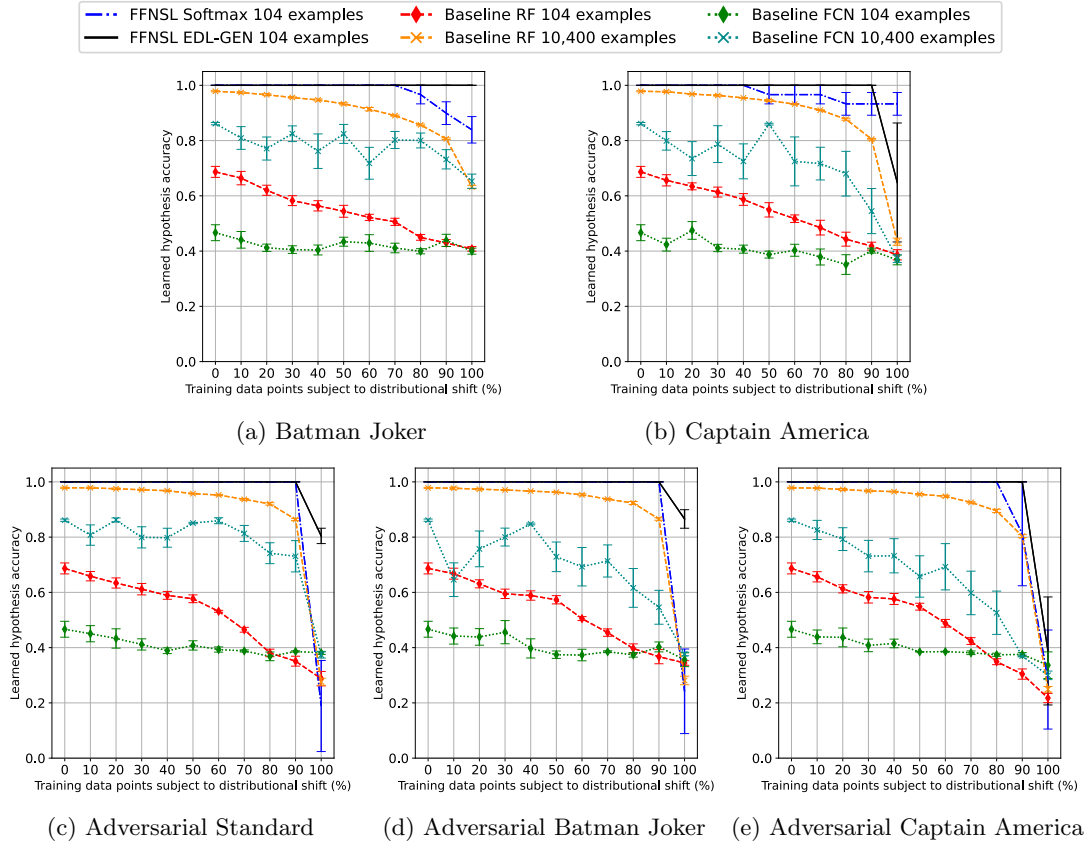


Figure 3.14: Accuracy of the learned hypotheses when increasing percentages of training data points are subject to distributional shifts, Follow Suit Winner task.

methodology outlined in this section.

3.4.3 Follow Suit Winner

The hypothesis learning dataset in the Follow Suit Winner domain is generated by simulating multiple games, where each game begins with a randomly shuffled deck of 52 playing cards, split between the four players. Each game consists of 13 tricks (since $\frac{52}{4} = 13$), and the card played by each player, along with the winner of each trick is stored. The *small* training sets contain 104 example tricks from 8 games, and the *large* training sets contain 10,400 example tricks from 800 games. A test set is created containing 1001 example tricks from 77 games. Figure 3.14 presents the accuracy of the learned hypotheses, when an increasing percentage of training data points are subject to distributional shifts.

Both FF-NSL instances outperform the baselines, learning far superior hypotheses when up to 90% of the training data points are subject to distributional shifts. The baselines require 100× the number of data points in order to attain similar performance. Despite this significant increase in data, FF-NSL still learns more accurate hypotheses. Figures 3.14a and 3.14b refer to the minor shifts. In these two cases, when the percentage of distributional shifts are very high (above 90%), the accuracy of the learned hypotheses decreases, but still remains between ~ 70 –100%, whereas the accuracy of the baselines trained with the same amount of data reduces to ~ 40 %. Figures 3.14c — 3.14e refer to the major shifts. In these cases, FF-NSL Softmax has a lower accuracy than in the minor shifts (see $> 90\%$), whereas FF-NSL EDL-GEN maintains a higher accuracy, on the Adversarial Standard, and Adversarial Batman Joker decks.

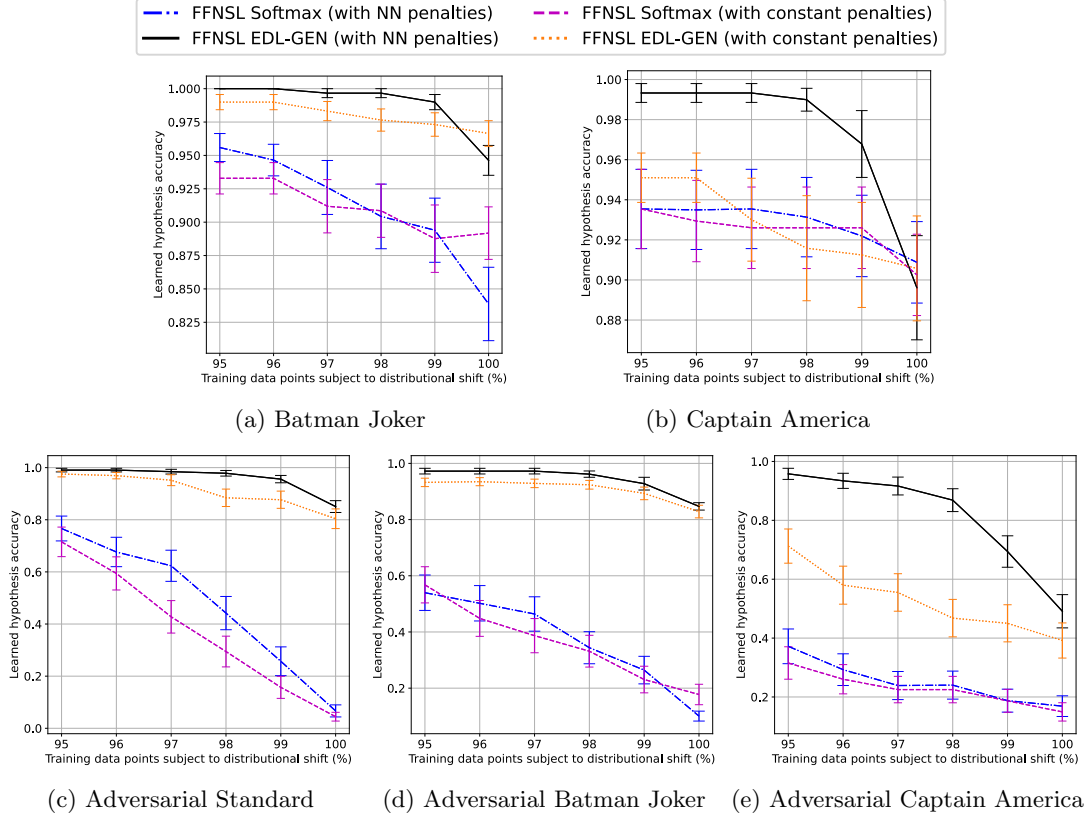


Figure 3.15: FF-NSL Softmax vs. FF-NSL EDL-GEN. Accuracy of the learned hypotheses with 95–100% distributional shifts, using 50 repeats. Follow Suit Winner task.

As Figure 3.14 presents a broad evaluation over 5 repeats, drawing conclusions as to the relative performance of both FF-NSL instances is challenging. Let us now investigate in more detail, whether FF-NSL EDL-GEN provides a performance benefit over FF-NSL Softmax, given the EDL-GEN network predicts with more informative confidence scores. For this analysis, we re-run the hypothesis learning tasks, but instead focus on high percentages of distributional shifts, 95–100%, as this is when FF-NSL instances deteriorate in terms of the accuracy of the learned hypothesis. We also run two additional FF-NSL instances, acting as baselines, where constant example weight penalties of the value 10 are used. This is to isolate the effect (if any) of the weight penalties calculated from the neural network confidence scores. Finally, we average the results over 50 experimental repeats (with 50 different randomly generated datasets), to generate statistically significant results, and enable us to draw more concrete conclusions. Figure 3.15 presents the results.

First, it is apparent that all methods perform better on the minor shifts, compared to the major shifts. Note the change in y-axis scale between each Figure. This is unsurprising, since the neural networks predict with higher accuracy for the minor shifts, as shown in Figure 3.7. Second, both of the EDL-GEN methods outperform both of the Softmax methods, and this relative performance difference is greater on the major shifts. An exception to this is the Captain America deck, where FF-NSL EDL-GEN with constant weight penalties performs similarly to both of the Softmax methods. Finally, when neural network confidence estimates are used to calculate the example weight penalties, the improved estimates of EDL-GEN only seem to provide a significant benefit on the Adversarial Captain America deck. As Softmax tends to always predict with high confidence (see Figure 3.7), this explains why the performance of FF-

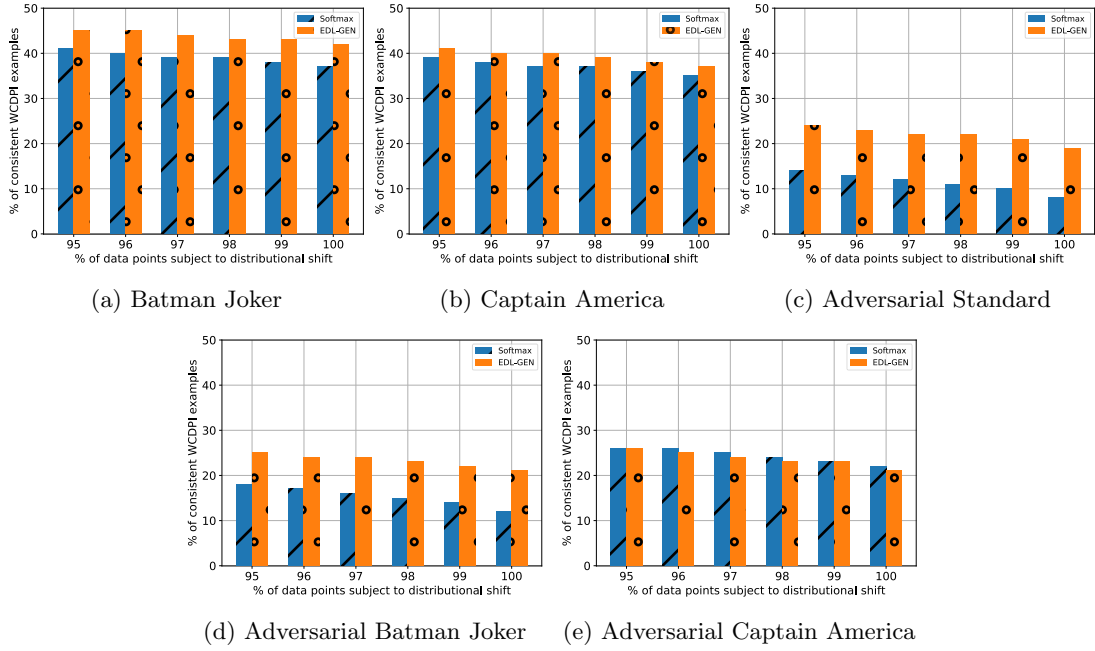


Figure 3.16: The effect of distributional shifts on the percentage of consistent WCDPI examples generated. Follow Suit Winner task.

NSL Softmax with neural network based penalties is similar to that with constant penalties. At this stage, three open questions remain: (1) Why do both EDL-GEN methods tend to outperform both Softmax methods on the major shifts, even though the neural networks predict with similar accuracy? (2) Why do the two FF-NSL approaches with constant weight penalties perform similarly on the Captain America deck, despite the EDL-GEN network predicting with higher overall accuracy? (3) Why does FF-NSL EDL-GEN with neural network based weight penalties significantly outperform the constant variant on the Adversarial Captain America deck, but not on the other decks?

Let us first consider question 1. A natural follow on question to ask is whether the EDL-GEN neural network leads to more examples with consistent concept predictions than Softmax, with respect to the downstream label, despite both networks having similar overall accuracy. This is important to investigate, because the LAS systems are capable of learning an accurate hypothesis from a few consistent examples. Therefore, we analyse the percentage of consistent WCDPI examples generated when 95–100% of the input data is subject to distributional shifts. For this analysis, we isolate the effect of the example weight penalties, and focus on comparing the performance difference between the two FF-NSL methods with constant penalties. The results are shown in Figure 3.16.

The EDL-GEN neural network leads to a higher number of consistent WCDPI examples, compared to that of Softmax, in all cases except Adversarial Captain America. Notice this difference is greater on the Adversarial Standard, and Adversarial Batman Joker decks, compared to the minor shifts. Given the relatively small number of WCDPI examples used by the symbolic learner (104), and the fact that the LAS symbolic learners are capable of learning a general hypothesis from very few consistent examples, this difference likely contributed to the gap in accuracy between FF-NSL EDL-GEN and FF-NSL Softmax in the Adversarial Standard, and Adversarial Batman Joker decks in Figure 3.15. However, on the Adversarial Captain America deck, both Softmax and EDL-GEN neural networks predict with similar accuracy, and

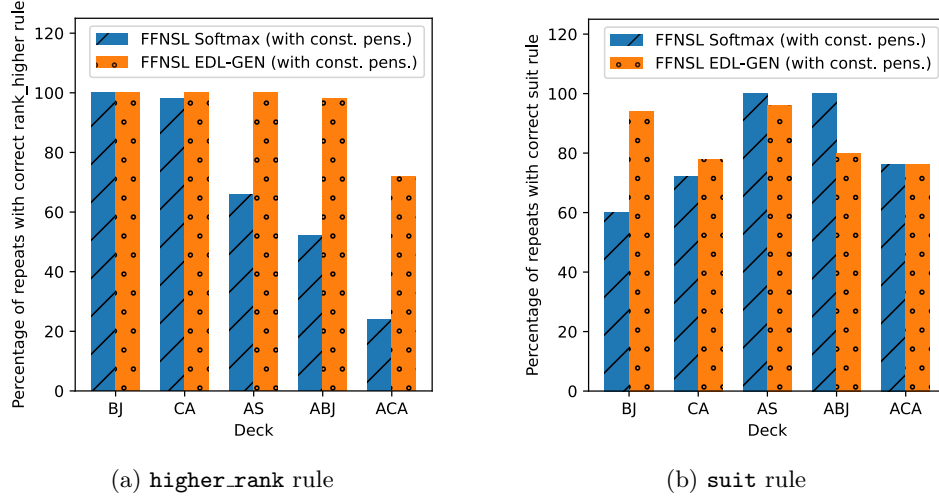


Figure 3.17: The percentage of experimental repeats that learned the correct follow suit rules, when 95% of input data points were subject to distributional shifts.

result in a similar percentage of consistent WCDPI examples, yet FF-NSL EDL-GEN clearly outperforms FF-NSL Softmax in Figure 3.15e, albeit not to the same degree as on the other major shifts. To investigate this further, we need to dive deeper, and analyse the specific rules learned in each case, as this directly influences the hypothesis accuracy. We therefore analyse the percentage of experimental repeats for both FF-NSL approaches with constant penalties that learn the correct **higher_rank** and **suit** rules, both of which are essential to solving the task successfully. The analysis is presented in Figure 3.17 when 95% of the input data points are subject to the distributional shift. The x-axis labels are the abbreviated names for each deck.

Focusing on the **higher_rank** rule, and the Adversarial Captain America deck (ACA) in Figure 3.17a, FF-NSL EDL-GEN learns the correct rule in 72% of experimental repeats, compared to 24% with FF-NSL Softmax. This helps to explain the performance difference between the two approaches in Figure 3.15e. There is also a greater difference in Figure 3.17a for the major shifts (AS, ABJ, ACA) compared to the minor shifts (BJ, CA), which also corresponds to the performance gaps in Figure 3.15. So, why does the symbolic learner fail to learn the correct **higher_rank** rule for 76% of the experimental repeats with FF-NSL Softmax, on the Adversarial Captain America deck? To answer this question, Figure 3.18c shows the distribution of the playing card rank predictions for the Softmax network. In this case, 60% of the predictions are of the rank “King”, which is almost the highest ranked card. Investigating the neural network card predictions within the generated WCDPI examples for this deck, only 9% of the examples contain a higher ranked card (an “Ace”), for the ground-truth winning player. This explains why FF-NSL Softmax performs poorly on this deck. There are barely any examples with a higher ranked card for the winning player, which itself explains why the symbolic learner often fails to learn the **higher_rank** rule. Without the **higher_rank** rule, the accuracy of the learned hypothesis is poor. Contrast this to FF-NSL EDL-GEN on the other hand, where 19% of the generated WCDPI examples contain a higher ranked card, since the distribution of card predictions is more even (see Figure 3.18d). This explains why the symbolic learner is able to learn the **higher_rank** rule more often, and therefore achieve a greater average for the learned hypothesis accuracy in Figure 3.15e. This, together with our analysis of the percentage of consistent WCDPI examples, answers our initial question 1, as to why both FF-NSL EDL-GEN methods tend to outperform both FF-NSL Softmax methods on the major shifts.

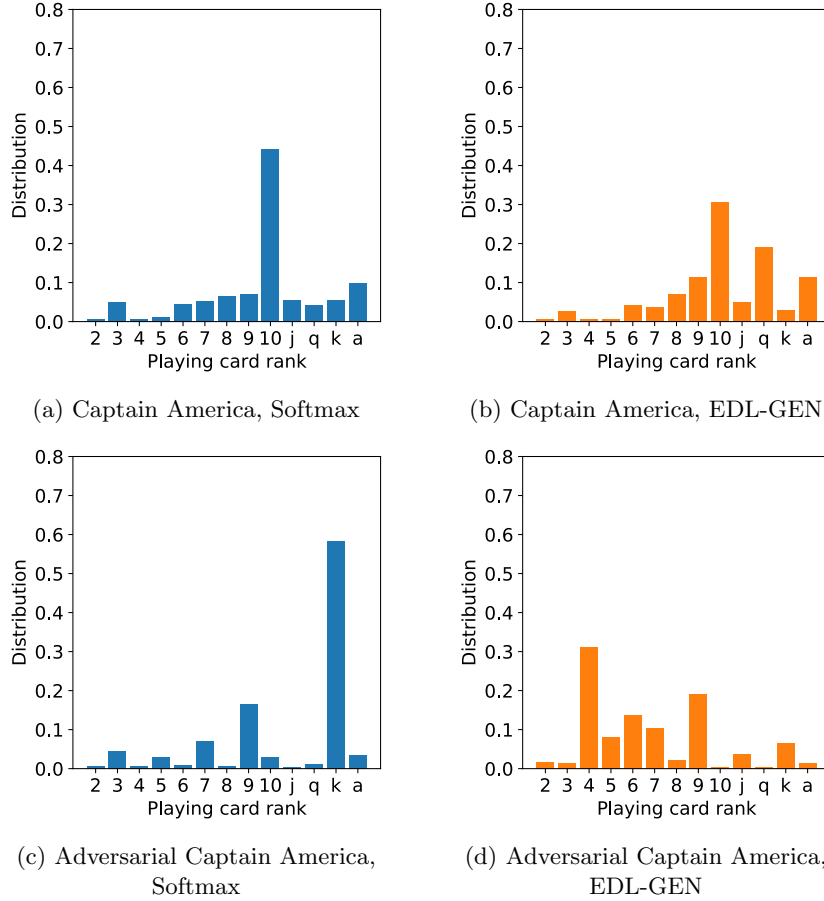


Figure 3.18: Distribution of playing card rank predictions for Softmax and EDL-GEN neural networks when the Captain America and Adversarial Captain America decks are used to apply distributional shift for 95% of input data points.

Let us now turn our attention to question 2, which is, *Why do the two FF-NSL approaches with constant weight penalties perform similarly on the Captain America deck, despite the EDL-GEN network predicting with higher overall accuracy?* Looking at the distribution of rank predictions in Figures 3.18a and 3.18b, it is clear both networks predict with a more even distribution. Although the softmax network often predicts rank “10”, 35% of the generated WCDPI examples contain a higher ranked card for the ground-truth winning player, since there are four more ranks higher than “10”. Considering FF-NSL EDL-GEN, 37% of the WCDPI examples contain a higher ranked card for the ground-truth winning player, which is very similar to FF-NSL Softmax. This explains why both FF-NSL variants learn the *higher_rank* rule in a similar percentage of repeats (see Figure 3.17a). This therefore explains why the performance of both variants with constant penalties is similar in Figure 3.15b, and answers our question 2.

Finally, we address the last remaining question, question 3: *Why does FF-NSL EDL-GEN with neural network based weight penalties significantly outperform the constant variant on the Adversarial Captain America deck, but not on the other decks?* To answer this, we plot the example weight penalty ratio introduced in Section 3.4, which measures the proportion of the total weight penalty allocated to correct WCDPI examples. The results are presented in Figure 3.19.

As expected, on all decks, the ratio of FF-NSL EDL-GEN with neural network confidence based weight penalties is higher than the corresponding variant with constant penalties. It is likely that due to the challenging nature of predicting accurately on the Adversarial Captain

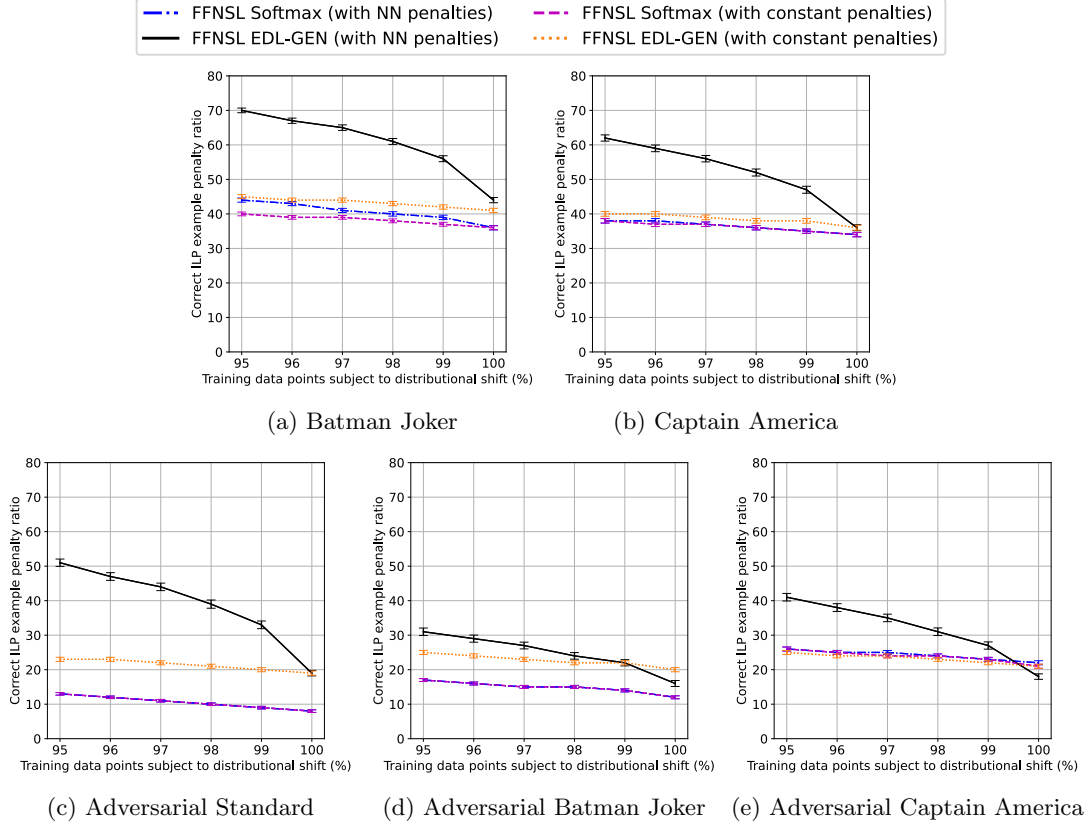


Figure 3.19: WCDPI example weight penalty ratio, 95–100% shifts. Follow Suit Winner task.

America deck, the slightly higher weight penalty ratio is enough to guide the symbolic learner to a better hypothesis, by encouraging it to focus on covering the examples with more accurate neural network predictions. Generally, it is also clear that the EDL-GEN confidence scores provide a benefit compared to Softmax, which instead performs very similar to the constant penalties. This is unsurprising since Softmax tends to always predict with high confidence. At 100% distributional shifts, the benefits of calculating weight penalties with the neural network confidence scores reduces, as there are very few consistent WCDPI examples with correct concept predictions. This explains why the gap between the hypothesis accuracy of FF-NSL EDL-GEN with neural network penalties, and that of FF-NSL EDL-GEN with constant penalties, reduces as the distributional shifts increase towards 100% (see Figure 3.15). This answers question 3.

In summary, a higher accuracy of the neural network predictions leads to a higher percentage (even if small) of consistent WCDPI examples, and better calibrated neural network confidence scores leads to an improved weight penalty ratio of correct WCDPI examples. Together, they provide an improved bias for the symbolic learner, which helps to guide the search for an optimal hypothesis. Let us now investigate the interpretability of the hypotheses learned. Figure 3.20 presents the results, where interpretability is measured in terms of the number of atoms in the learned hypothesis.

FF-NSL learns significantly more interpretable knowledge than the baseline approaches (note the logarithmic scale on the y-axis), especially when the baselines are trained with $100\times$ the amount of data. These models reach high accuracy at the expense of much lower interpretability. This is because they learn a more complex mapping between input and output, instead of learning general rules, as is the case for our approach. The FCN trained with the same amount of data as ours, has similar interpretability, because the model learns to largely predict the same class,

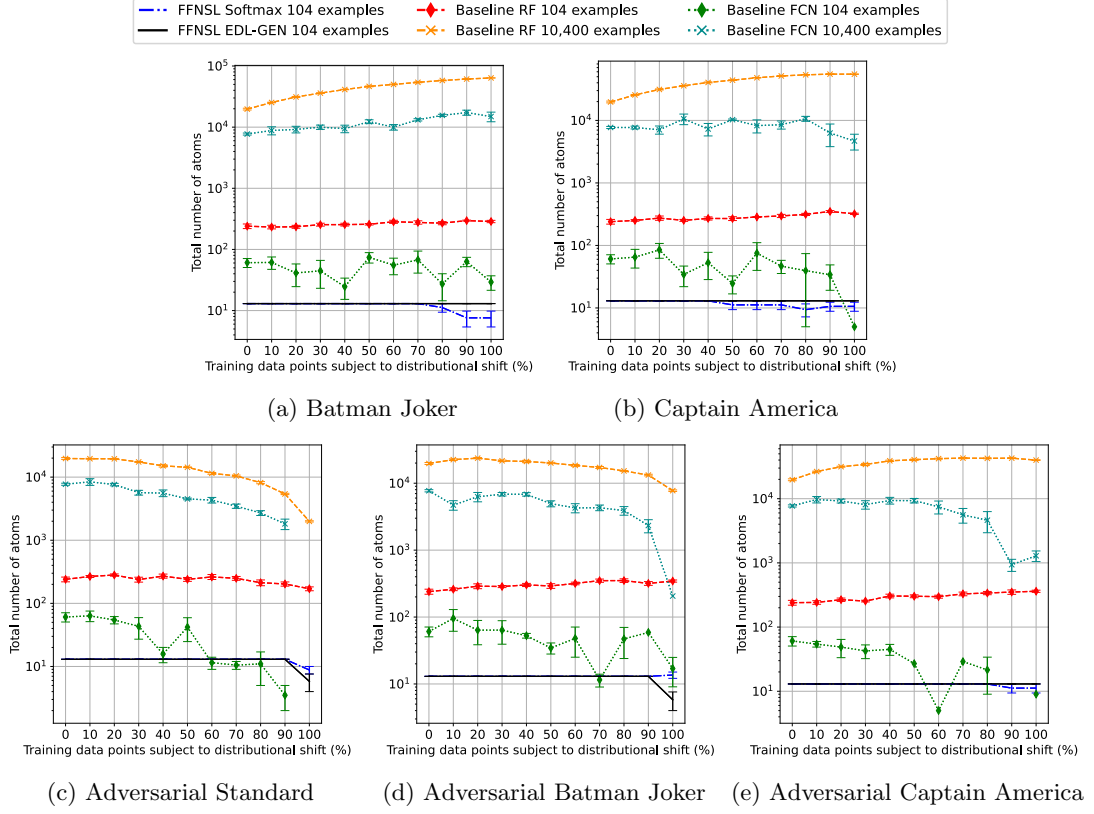


Figure 3.20: Interpretability of the learned hypotheses, Follow Suit Winner task.

and therefore the surrogate decision tree is very small. This is reflected in the poor performance of the FCN shown in Figure 3.14. To qualitatively demonstrate the interpretability of FF-NSL, we now present a selection of learned hypotheses with and without distributional shifts applied during training. When no distributional shifts are applied, i.e., at points 0% on the x-axes in Figure 3.14, the following hypothesis is learned:

```

winner(X) :- not p1(X), player(X).
p1(V1) :- V2 != V3; suit(1,V2); suit(V1,V3); player(V1); suit(V2); suit(V3).
p1(V1) :- higher_rank(V2,V1); suit(1,V3); suit(V2,V3); player(V1); player(V2); suit(V3).

```

The first rule states that player X is a winner if *neither* of the bottom two rules hold. The second rule holds if the suit of player X is different to the suit of player 1, and the final rule holds if there is another player with a higher ranked card with the same suit as player 1. The **player**, **suit** and **higher_rank** predicates are defined in the background knowledge (see Appendix A). In the case of 100% shifts, consider the Batman Joker deck in Figure 3.14a. The following hypothesis is learned on one experimental repeat:

```

winner(X) :- not p1(X), player(X).
p1(V1) :- higher_rank(V2,V1); player(V1); player(V2).

```

In this hypothesis, a player is the winner if they have played the highest ranked card. The symbolic learner fails to learn the rule regarding the suit having to match the suit of player 1. Finally, to investigate the scalability of FF-NSL, we also record the time required to learn a hypothesis. Note the time required to train the neural network is excluded, since it is the same for all approaches. The results are shown in Figure 3.21.

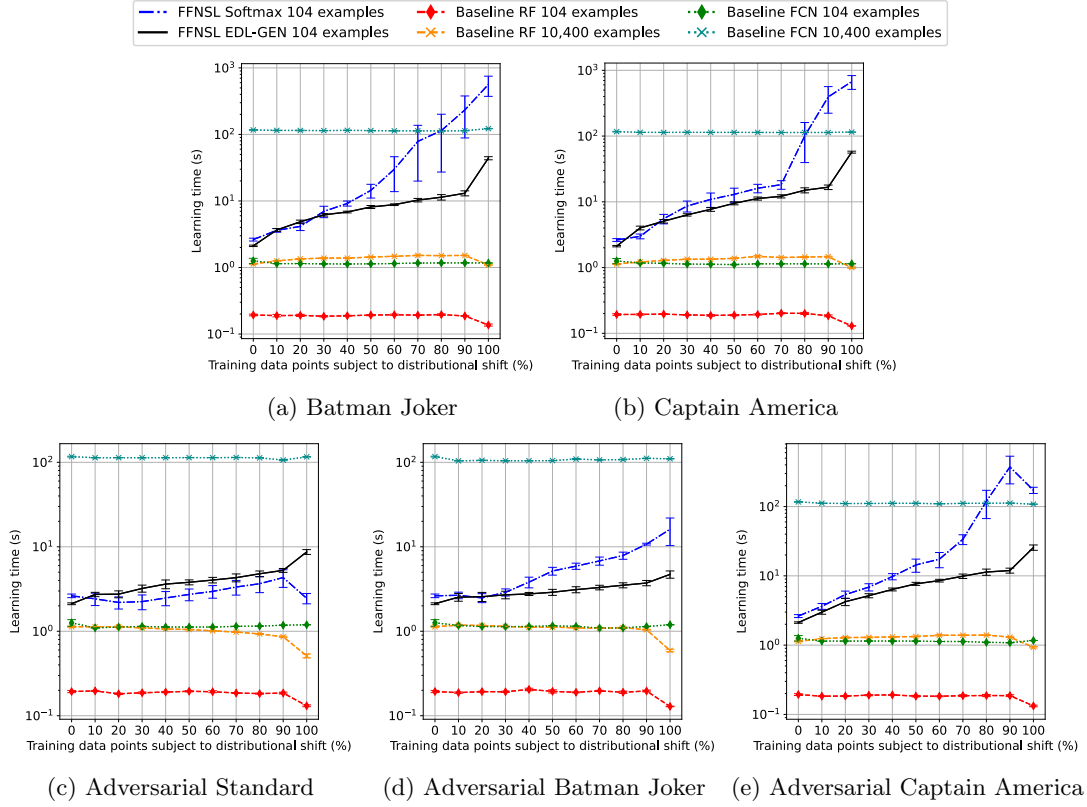


Figure 3.21: Learning time. Follow Suit Winner task.

Both FF-NSL approaches learn with an order of magnitude of time similar to that of the FCN, when trained with the same number of examples, and no distributional shifts are applied. As the shifts increase, FF-NSL takes longer because the symbolic learner ILASP requires more iterations to prove optimality with respect to minimising the total penalty on the examples. However, the learning time of FF-NSL EDL-GEN does not increase as quickly, when compared to FF-NSL Softmax. This is because the example weight penalties are much more informative (see Figure 3.19) and ILASP requires fewer iterations to prove optimality.

In conclusion, our analysis of the learned hypotheses shows that FF-NSL outperforms the baseline approaches in terms of accuracy and interpretability, even when the baselines are trained with $100\times$ the amount of data. FF-NSL EDL-GEN outperforms FF-NSL Softmax, as the EDL-GEN neural network predictions are more accurate, and in the case of major distributional shifts, the uncertainty-aware confidence scores generates more informative weight penalties for the symbolic learner. Finally, we have shown that more informative weight penalties result in faster hypothesis learning times for the symbolic learner, when the iterative ILASP system is used.

3.4.4 Sudoku Grid Validity

Having evaluated in detail the performance of the learned hypotheses on the Follow Suit Winner task, we now explore the Sudoku Grid Validity task, where we consider two cases: 4×4 , and 9×9 Sudoku grids. The purpose of these tasks is to evaluate FF-NSL, when given a larger number of images in each input sequence X , than observed in the Follow Suit Winner task. Valid Sudoku data is obtained from a puzzle generator,³ which contains 4–6 images in each 4×4 grid, and

³<https://www.menneske.no/sudoku/2/>

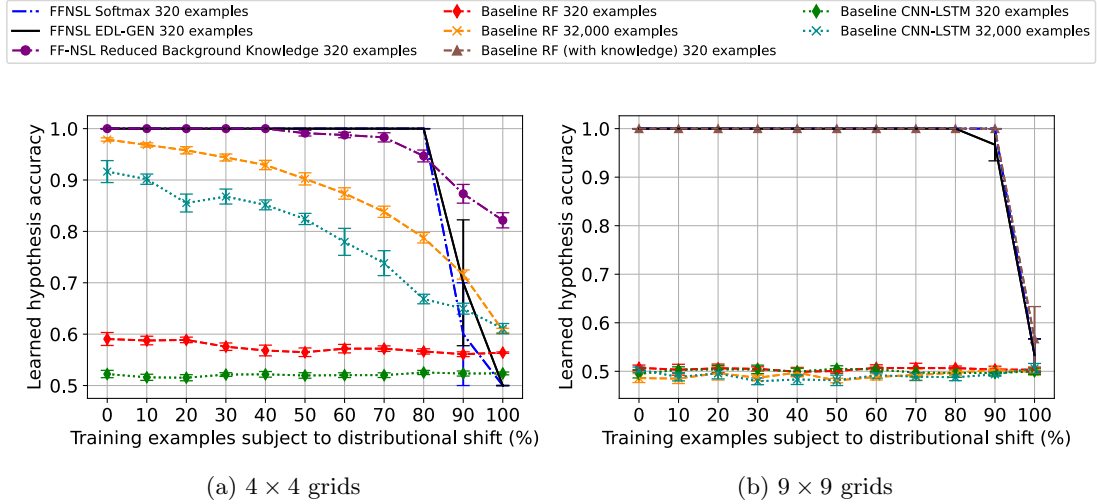


Figure 3.22: Learned hypothesis accuracy with increasing percentages of data points subject to distributional shifts, Sudoku Grid Validity task.

21–36 images in each 9×9 grid. When distributional shifts are applied to image sequences with a greater number of images, this becomes challenging for the symbolic learner, as each generated example contains more concepts that are likely to be predicted incorrectly. Invalid data is generated by taking a valid example from the puzzle generator that does not exist in our curated set of valid data points, and changing one digit at random in a row, column or block to match another digit in the same row, column or block. All invalid datasets contain an equal distribution of data points containing two of the same digit in a row, column or block. The *small* training sets contain 320 data points, each consisting of 160 *valid* and 160 *invalid* grids. The *large* training sets contain 32,000 data points, with 16,000 valid and 16,000 invalid data points. Finally, separate test sets are created for 4×4 and 9×9 grids, which contain 1000 data points: 500 valid and 500 invalid. Figure 3.22 presents the accuracy of the hypotheses learned when an increasing percentage of training data points are subject to distributional shifts.

All FF-NSL approaches outperform the baselines in both 4×4 and 9×9 tasks, learning far more accurate hypotheses. In the 4×4 task, the baselines required $100\times$ the amount of data to reach an accuracy close to that of FF-NSL, whereas in the 9×9 task, the baselines fail completely. In Figure 3.22a the purple line is FF-NSL EDL-GEN with the explicit background knowledge about the Sudoku grid removed. In this case, the symbolic learner has to infer the grid structure, given a more general notion of division and cell coordinates as meta-data. This is a significantly harder learning task, yet FF-NSL still outperforms the baselines. This variant also outperforms the original FF-NSL approaches when explicit background knowledge about the Sudoku grid is provided, in the cases of 90% and 100% distributional shifts. This is because the symbolic learner is less constrained by the explicit knowledge of the grid, and alternative hypotheses can be learned which better accommodate the (incorrect) predictions of the neural networks. Given explicit knowledge about the grid, the hypothesis space contains rules that perform either very well or very poorly. The background knowledge and mode declarations for the symbolic learner are listed in full in Appendix A. In Figure 3.22b, the brown line shows the accuracy of the RF with the 3 Boolean input features, post-processed from the pre-trained neural network predictions. These features indicate if any two digit values appear in the same row, column or block, which effectively encodes the Sudoku grid knowledge into the RF learning task. This constitutes even more background knowledge than what was provided to our FF-NSL

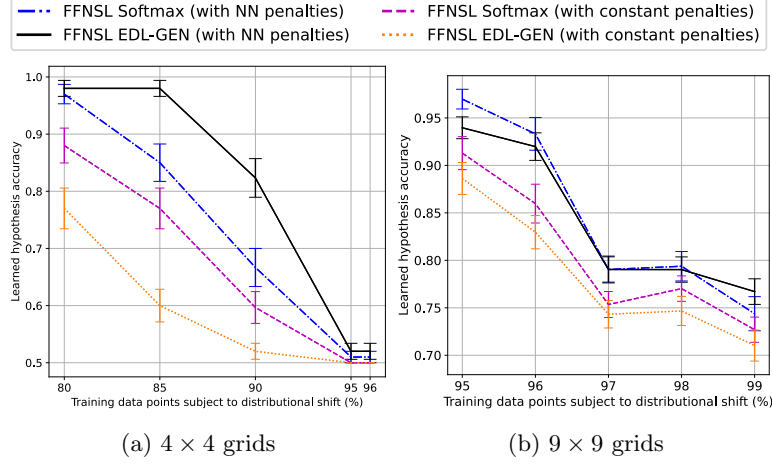


Figure 3.23: FF-NSL Softmax vs. FF-NSL EDL-GEN. Average accuracy of learned hypotheses over 50 repeats. Sudoku Grid Validity task.

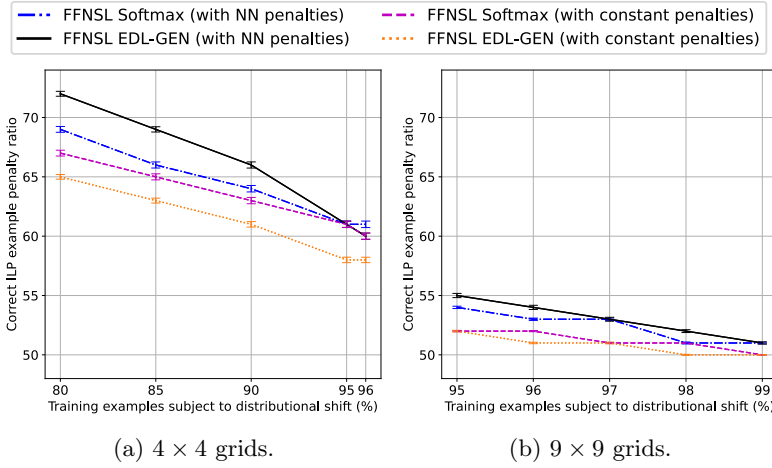


Figure 3.24: WCDPI example weight penalty ratio, Sudoku Grid Validity task.

approaches, and FF-NSL performs similarly to this baseline that uses additional knowledge.

Let us now investigate whether using the EDL-GEN neural network provides a benefit over Softmax in the presence of high percentages of distributional shifts. We focus on 80–96% shifts for the 4×4 Sudoku Grid Validity task and 95–99% shifts for the 9×9 task, as this is where the performance of FF-NSL deteriorates. Similarly to the Follow Suit Winner task, we run 50 experimental repeats, and also two additional baseline FF-NSL approaches with constant weight penalties. Figure 3.23 presents the results. First, in both 4×4 and 9×9 grids, the Softmax and EDL-GEN FF-NSL approaches that used example weight penalties calculated from neural network confidence scores outperformed the approaches with constant weight penalties.

To investigate this further, we explore the WCDPI example weight penalty ratio for the 4×4 and 9×9 tasks. Figure 3.24 shows that both Softmax and EDL-GEN FF-NSL approaches with neural network penalties, have a larger weight penalty ratio than the corresponding approaches with constant penalties. For FF-NSL EDL-GEN, this is expected, since the neural network confidence scores are better calibrated with respect to the uncertainty for out-of-distribution data. On these tasks, the Softmax neural network also had a slightly more varied confidence score distribution over data subject to distributional shift (see Figure 3.9), which explains the

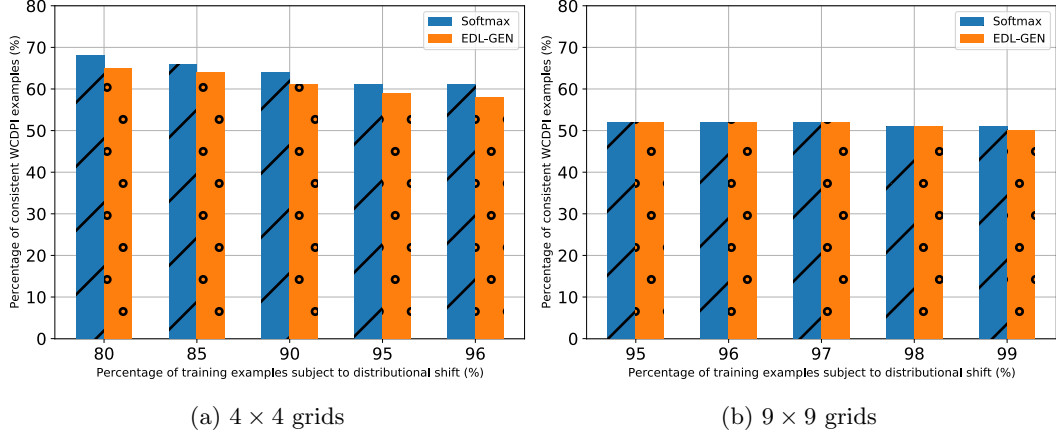


Figure 3.25: The effect of applying distributional shifts on the percentage of consistent WCDPI examples, Sudoku Grid Validity task.

higher weight penalty ratio compared to using constant penalties. These differences explain the performance gain of FF-NSL with neural network based weight penalties versus constant penalties in Figure 3.23. Similarly, the difference between the neural network based example weight penalty ratio of FF-NSL EDL-GEN and FF-NSL Softmax, also explains why FF-NSL EDL-GEN outperforms FF-NSL Softmax in Figure 3.23a for 4×4 grids. For the 9×9 grids, the difference between the neural network based example weight penalty ratio of FF-NSL EDL-GEN and FF-NSL Softmax, is very small, which explains why FF-NSL EDL-GEN and FF-NSL Softmax show a similar performance in Figure 3.23b. Also, the Softmax neural network is slightly more accurate when predicting standard MNIST digits within the distribution of the training data, which explains why the performance of FF-NSL Softmax is slightly higher than FF-NSL EDL-GEN in Figure 3.23 at 95% shift. FF-NSL Softmax ends up worse than FF-NSL EDL-GEN at 99% shift, since the EDL-GEN neural network is more accurate given out-of-distribution data. Now, in Figure 3.23a, FF-NSL Softmax with constant weight penalties outperforms FF-NSL EDL-GEN with constant weight penalties, whereas in Figure 3.23b these two approaches perform similarly. To investigate this further, we consider the percentage of consistent WCDPI examples for the symbolic learner, when distributional shifts are applied. The results are shown in Figure 3.25.

For the 4×4 task, despite the EDL-GEN neural network predicting on average more accurately (see Figure 3.9), a lower percentage of consistent WCDPI examples are generated than when the Softmax neural network is used. This explains the lower performance in Figure 3.23a of FF-NSL EDL-GEN with constant penalties. Using the WCDPI example weight penalties calculated from EDL-GEN neural network confidence scores rectifies this issue, and the symbolic learner is biased to learn a hypothesis from WCDPI examples containing correct neural network predictions. For the 9×9 task, both neural networks lead to a similar percentage of consistent WCDPI examples, which explains the similar performance of FF-NSL EDL-GEN and FF-NSL Softmax with constant penalties, shown in Figure 3.23b. Let us now investigate the interpretability of FF-NSL compared to the baseline approaches. The results are shown in Figure 3.26.

As observed in the Follow Suit Winner task, FF-NSL learns significantly more interpretable hypotheses than the baseline approaches. Let us now present example hypotheses learned with and without distributional shifts applied. For the 4×4 and 9×9 grid Sudoku tasks, when no distributional shifts are applied (i.e., at points 0% on the x-axes in Figure 3.22), the following

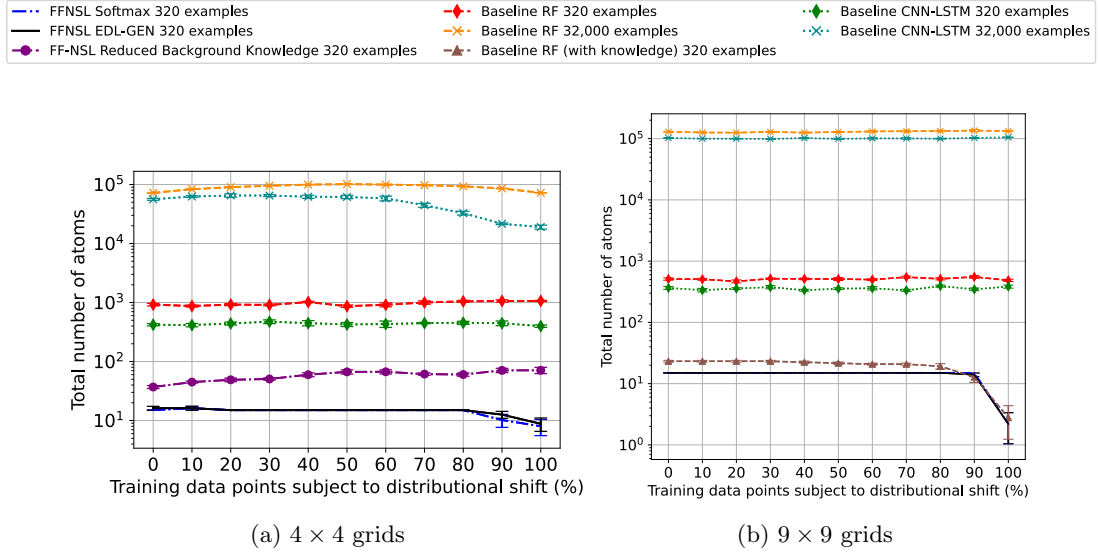


Figure 3.26: Interpretability of the learned hypotheses, Sudoku Grid Validity task.

hypothesis is learned, which states that a Sudoku grid is invalid if there are two of the same digits in a row, column, or block:

```
invalid :- neq(V1,V0), digit(V0,V2), digit(V1,V2), row(V0,V3), row(V1,V3).
invalid :- neq(V1,V0), digit(V0,V3), digit(V1,V3), col(V0,V2), col(V1,V2).
invalid :- neq(V2,V1), digit(V1,V3), block(V2,V0), block(V1,V0), digit(V2,V3).
```

The `neq`, `digit`, `block`, `row` and `col` predicates are defined in the background knowledge (see Appendix A). For the 4×4 grid task, when 100% of the training data points are subject to distributional shifts, the following hypothesis is learned:

```
invalid :- neq(V2,V1), neq(V3,V1), neq(V3,V2), block(V2,V0), block(V3,V0), block(V1,V0).
invalid :- not block(V1,V0), block(V2,V0), col(V2,V3), col(V1,V3).
invalid :- not block(V2,V0), block(V1,V0), row(V1,V3), row(V2,V3).
```

The first argument in the `block`, `row` and `col` predicates is a string representing cell coordinates, and the second argument is an identifier (e.g., block 1, block 2, etc...). The first rule states that a grid is invalid if there are three cells in the same block that have different coordinates. The second rule states that a grid is invalid if there are two cells within the same column that are in different blocks, and the third rule states that a grid is invalid if there are two cells within the same row that are in different blocks. Therefore, this hypothesis always returns invalid at test-time, hence the 0.5 accuracy in Figure 3.22 at 100% shifts. As for the learning time, the results are shown in Figure 3.27.

In both 4×4 and 9×9 tasks, the learning time for FF-NSL does not increase as the percentage of input data subject to distributional shifts increases. This is because FastLAS learns a hypothesis by solving an optimisation problem with respect to all generated WCDPI examples. This is not the case in the Follow Suit Winner task, where the ILASP system learns an optimal hypothesis iteratively. It is interesting to note that in Figure 3.27a, FF-NSL’s learning time has the same order of magnitude as that of the CNN-LSTM trained with $100\times$ the amount of data, which has lower accuracy up to 90% of distributional shifts.

In conclusion, we have shown that for the learned hypotheses evaluation, FF-NSL outperforms the baseline approaches in terms of accuracy and interpretability, even when the baselines are trained with $100\times$ the amount of data. Furthermore, in this task, WCDPI example weight penalties have a larger impact on the performance of FF-NSL. We have also shown that FF-NSL

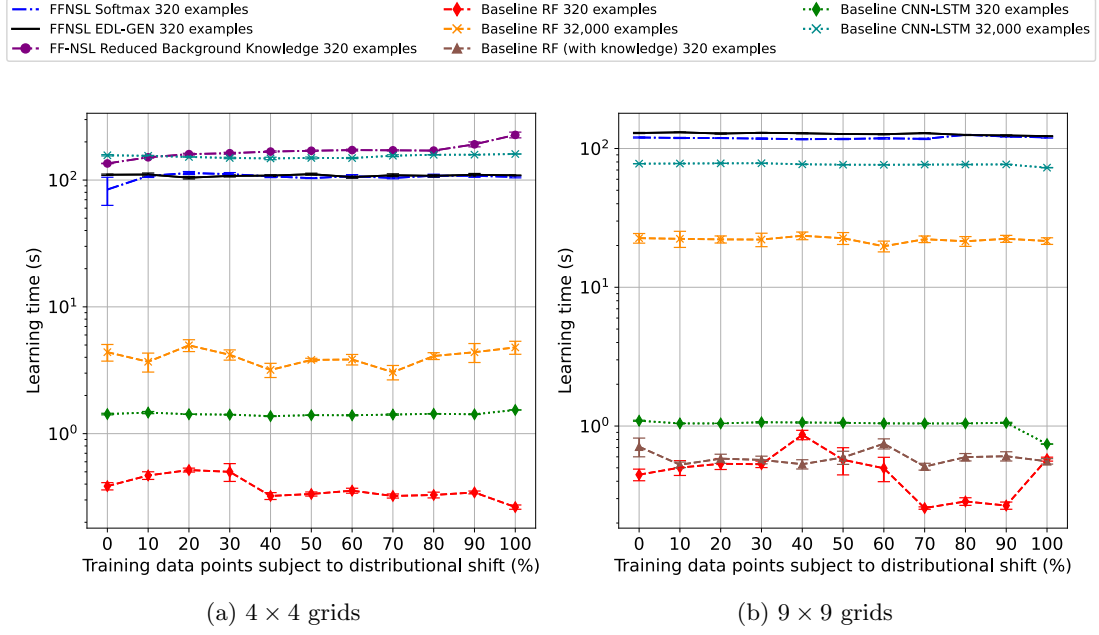


Figure 3.27: Hypothesis learning time, Sudoku Grid Validity task.

can scale to learning hypotheses where many more images are observed in each input data point, and finally, FF-NSL learns in a timely manner. Let us now present the accuracy of the learned hypotheses for the real-world datasets.

3.4.5 Real-world Datasets

Figure 3.28 shows the accuracy of the learned hypotheses in each task, when an increasing percentage of labelled unstructured data are subject to distributional shifts.

In the Crop Yield Prediction task, both instances of FF-NSL learn accurate hypotheses, until 90% of the data are subject to a distributional shift, and outperform all of the baseline approaches, even when the baselines are trained with significantly more data. In the Indoor Scene Classification task, FF-NSL also outperforms the baseline approaches, learning the correct hypotheses when up to 50% of the data are subject to a distributional shift. Since there are fewer baselines in this task, we also include the results where FF-NSL uses constant weight penalties for the examples for the symbolic learner. Setting the weight penalties for the examples based on the neural network confidence scores, leads to more accurate hypotheses, compared to using constant penalties. This is because using the neural network-based weight penalties enables FastLAS to find a better optimal solution within the 10 minute timeout, as the more informative weight penalties give a clearer optimisation signal for the final solving stage. With constant weight penalties, the optimisation takes significantly longer, as the distributional shift increases (see Figure 3.31b below). Now, we also run the same weight penalty analysis for the Crop Yield Prediction task. We run 50 experimental repeats between 95–100% shifts. Figure 3.29 presents the results.

The EDL-GEN instances of FF-NSL clearly outperform the Softmax FF-NSL instances in terms of the accuracy of the learned hypothesis (see Figure 3.29a). This is because the EDL-GEN neural network predicts with greater accuracy than Softmax, when the distributional shift is applied (see Figure 3.11). However, setting the weight penalties of the examples for the symbolic learner based on neural network confidence scores made very little difference compared

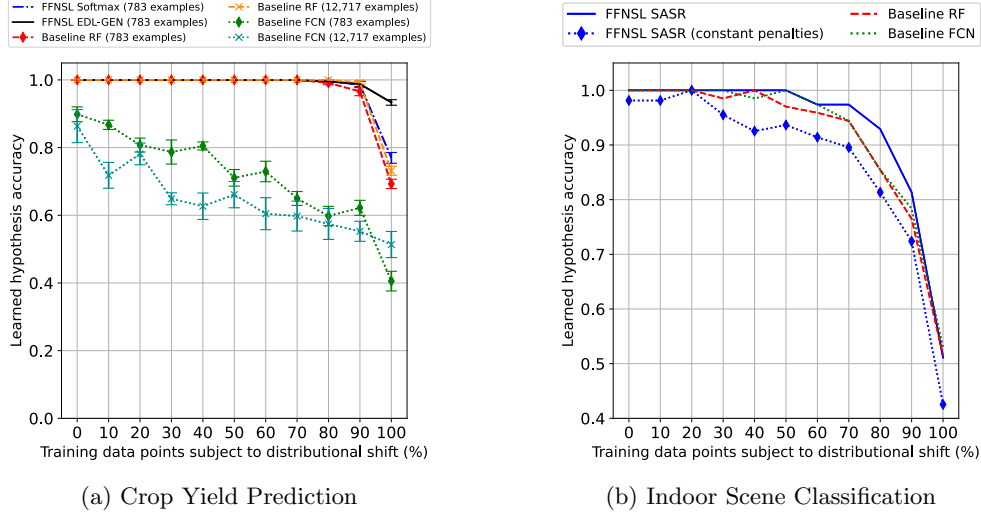


Figure 3.28: Learned hypothesis accuracy.

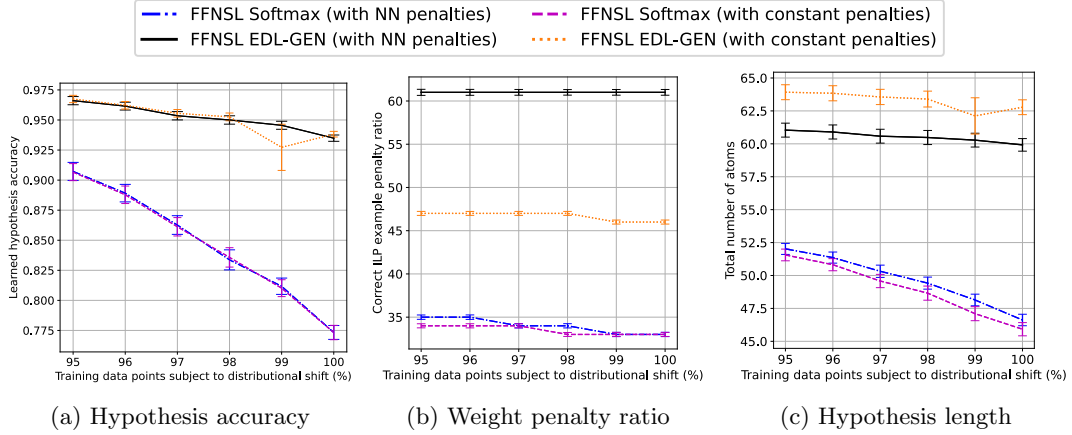


Figure 3.29: The effect of setting WCDPI example weight penalties based on neural network confidence scores, compared to constant penalties, Crop Yield Prediction task. 95-100% shifts, 50 repeats.

to using constant penalties. Therefore, we investigate the weight penalty ratio (Figure 3.29b). As expected, both FF-NSL Softmax instances had a similar weight penalty ratio, due to the Softmax neural network predicting with high confidence when distributional shift is applied (see Figure 3.11). The EDL-GEN instances do however show a difference, and the neural network-based weight penalties do provide a more informative signal. The question therefore, is why did this not translate into an improvement in the learned hypothesis accuracy? It turns out that the benefit is realised in the length of the learned hypothesis (Figure 3.29c), as FF-NSL EDL-GEN with neural network weight penalties learns a shorter hypothesis than when constant penalties are used. Comparing Figure 3.29c to Figure 3.29a, you can see that at 99% shifts, when the accuracy of FF-NSL EDL-GEN with constant penalties decreases, the length of the learned hypotheses also decreases, whilst FF-NSL EDL-GEN with neural network weight penalties achieves a higher accuracy with a shorter hypothesis. With constant penalties, to account for the level of noise, the symbolic learner has to learn more rules that map additional values of location type, plant species and disease to crop yield, in order to maintain the same level of accuracy as when neural network-based penalties are used. Figure 3.30 presents the interpretability results for both tasks.

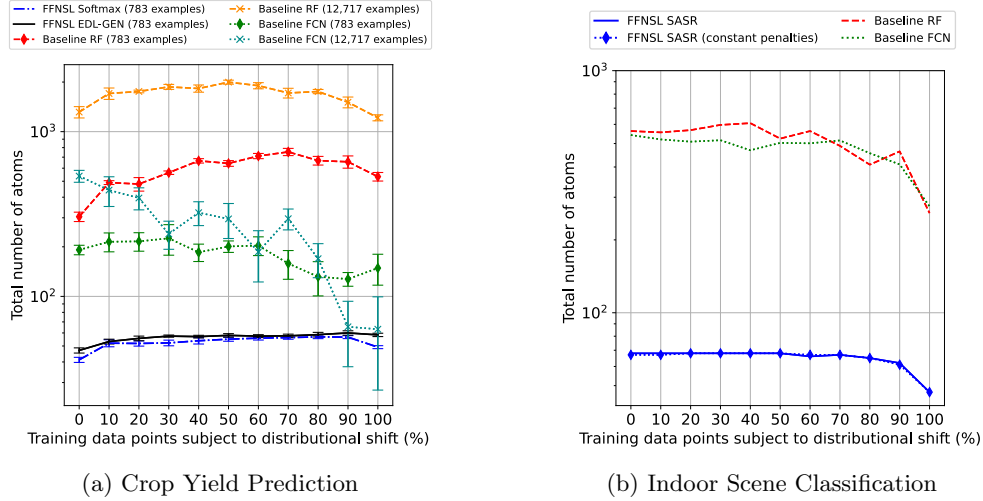


Figure 3.30: Interpretability of the learned hypotheses. Real world datasets.

FF-NSL learns significantly more interpretable hypotheses than the baseline approaches. Here are the rules learned in the Crop Yield Prediction task, when no distributional shift is applied:

```
yield(0) :- disease(bacterial_spot), location(18).
yield(0) :- disease(black_rot), location(18).
yield(1) :- species(corn), location(7).
yield(1) :- disease(leaf_blight_isariopsis_leaf_spot_).
yield(2) :- disease(late_blight), location(6).
yield(2) :- disease(leaf_scorch).
...
```

These rules identify the particular class of yield for certain species, diseases, and locations. For the Indoor Scene Classification task, the learned rules follow a similar format:

```
label(0) :- image(shoeshop).
label(1) :- image(nursery).
label(2) :- image(inside_subway).
label(3) :- image(bowling).
label(4) :- image(meeting_room).
...
```

where the super-class label is identified for each indoor scene image. In the case of distributional shifts, some species, diseases and locations are assigned to the incorrect yield type in the Crop Yield Prediction task, and in the Indoor Scene Classification task, some scene level image descriptions are assigned to the incorrect super-class.

Finally, 3.31 presents the learning time results. FF-NSL learns a hypothesis faster than the baselines trained with more examples in the Crop Yield Prediction task, and is slower than the baselines in the Indoor Scene Classification task. Figure 3.31b clearly shows the computational benefit of setting WCDPI example weight penalties based on neural network confidence scores, as a hypothesis is learned significantly faster than when constant penalties are used. The near constant learning times at 80–100% shifts for FF-NSL SASR with neural network-based penalties, and 30–100% shifts with constant penalties, is because we implemented a timeout for FastLAS to return the most optimal hypothesis found after 10 minutes, in order to obtain results in a timely manner.

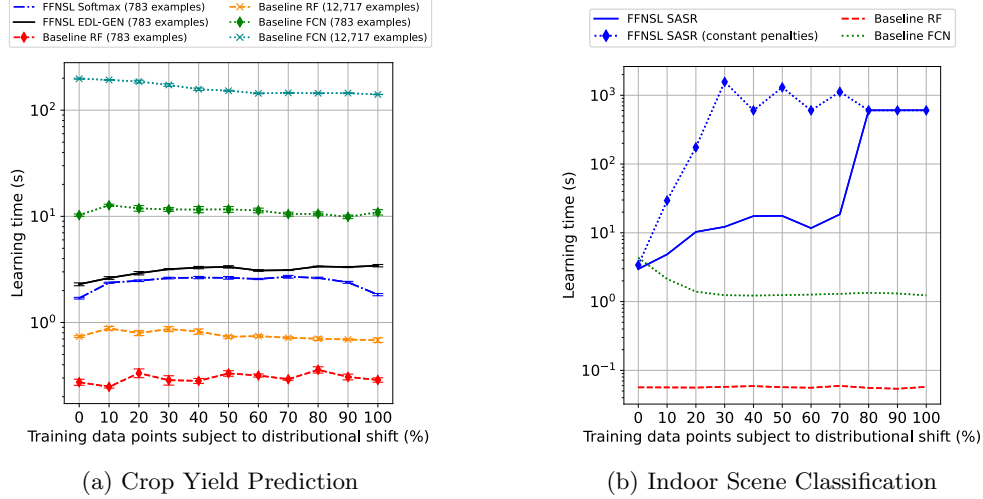


Figure 3.31: Hypothesis learning time. Real world datasets.

3.5 RQ2: Evaluating the FF-NSL Framework

In this section, we evaluate the *inference* capability of FF-NSL, where the pre-trained neural network(s), together with the learned hypothesis, are used to make a downstream prediction given *unseen* images and meta-data. This evaluation is achieved using a held-out test set, where each example contains a previously unseen image sequence X , associated meta-data M , together with a downstream label y . We extract concepts from the raw images in X using the learned perception function g , and encode the predictions, alongside the associated meta-data, the learned hypothesis, and any background knowledge into an ASP program. The ASP program is then solved using Clingo (Gebser et al., 2019) (an ASP solver), which outputs a downstream prediction. This is compared to the ground-truth label, and an accuracy score is computed. To assist the evaluation, and to provide insight into where mistakes are being made, we evaluate the neural network accuracy in extracting the concepts from the images in X , with respect to ground truth concept labels. This enables us to identify whether any downstream classification error is due to the neural network, the learned hypothesis, or both. We also evaluate FF-NSL under distributional shifts. We inject the same percentages of distributional shifts observed during symbolic learning into the held-out test set, and evaluate the overall accuracy in predicting the downstream label correctly. This evaluates the performance of FF-NSL in realistic scenarios where distributional shifts occur during learning and inference. Let us now present the results for each problem domain.

3.5.1 Follow Suit Winner

Figure 3.32 presents the accuracy of the entire FF-NSL framework when evaluated over a test set subject to the same types of distributional shifts as observed during training. FF-NSL outperforms the baselines trained with the same amount of data at each percentage of distributional shift on the Batman Joker deck, and until $\sim 80\%$ distributional shifts on the other decks. The baselines require $100\times$ the amount of data in order to match or outperform FF-NSL. The baselines trained with the same amount of data outperform FF-NSL for $> 80\%$ distributional shifts. This is because they largely predict player 1 which is sufficient to reach $\sim 40\%$ accuracy on the test set: the Follow Suit Winner task is biased towards player 1 because winning depends on playing the highest ranked card with the same suit as player 1. In the test set, 38.6% of

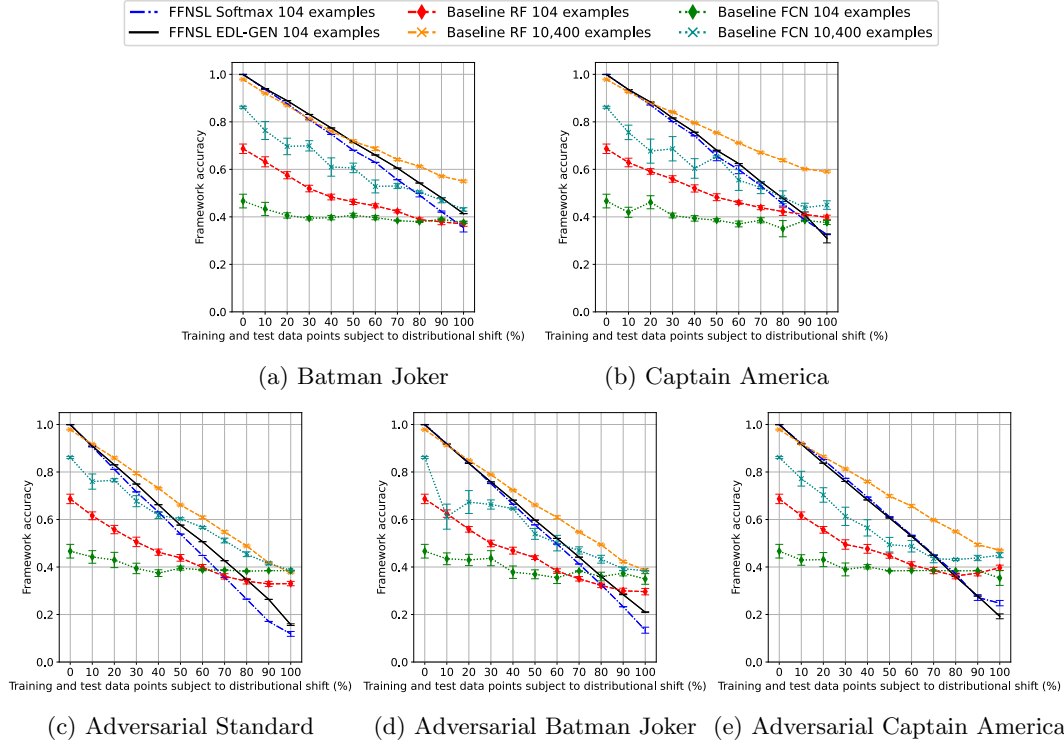


Figure 3.32: Accuracy of the FF-NSL framework when training and test data were subject to the same % of distributional shifts. Follow Suit Winner.

the data is indeed labelled with player 1 as the winner, which roughly corresponds to the performance of the baselines trained with the same number of examples at 100% shifts. FF-NSL EDL-GEN outperforms FF-NSL Softmax because of two reasons. First, the rules learned by FF-NSL EDL-GEN in the presence of high percentage of distributional shifts, are more accurate, due to the higher number of consistent WCDPI examples generated, when the EDL-GEN neural network is used. Second, the EDL-GEN neural network provides a more informative bias to the symbolic learner, through better calibrated example weight penalties. Finally, the decrease in performance of the FF-NSL approaches appears to be linear given an increase in the percentage of examples subject to distributional shifts. This is primarily due to the accuracy of the neural network predictions. Figure 3.33 shows that indeed the accuracy of neural network predictions over unseen card images decreases linearly, as the percentage of distributional shifts increase.

As shown in Figure 3.33, the EDL-GEN neural network is more accurate than Softmax in predicting unseen playing card images, in the case of minor shifts, although it is the same with major shifts. FF-NSL EDL-GEN’s better performance on the Batman Joker deck in Figure 3.32 is somewhat influenced by the more accurate neural network predictions, but generally the performance across all decks depends on the accuracy of the learned hypothesis, and also the types of prediction mistakes the neural network makes during inference (i.e., the distribution of rank values, see Figure 3.18).

3.5.2 Sudoku Grid Validity

Figure 3.34 presents the results of both Sudoku Grid Validity tasks.

In the 4×4 task, FF-NSL outperforms the baselines until 80% of the test data are subject to distributional shifts, even when the baselines are trained with $100\times$ the amount of data. In

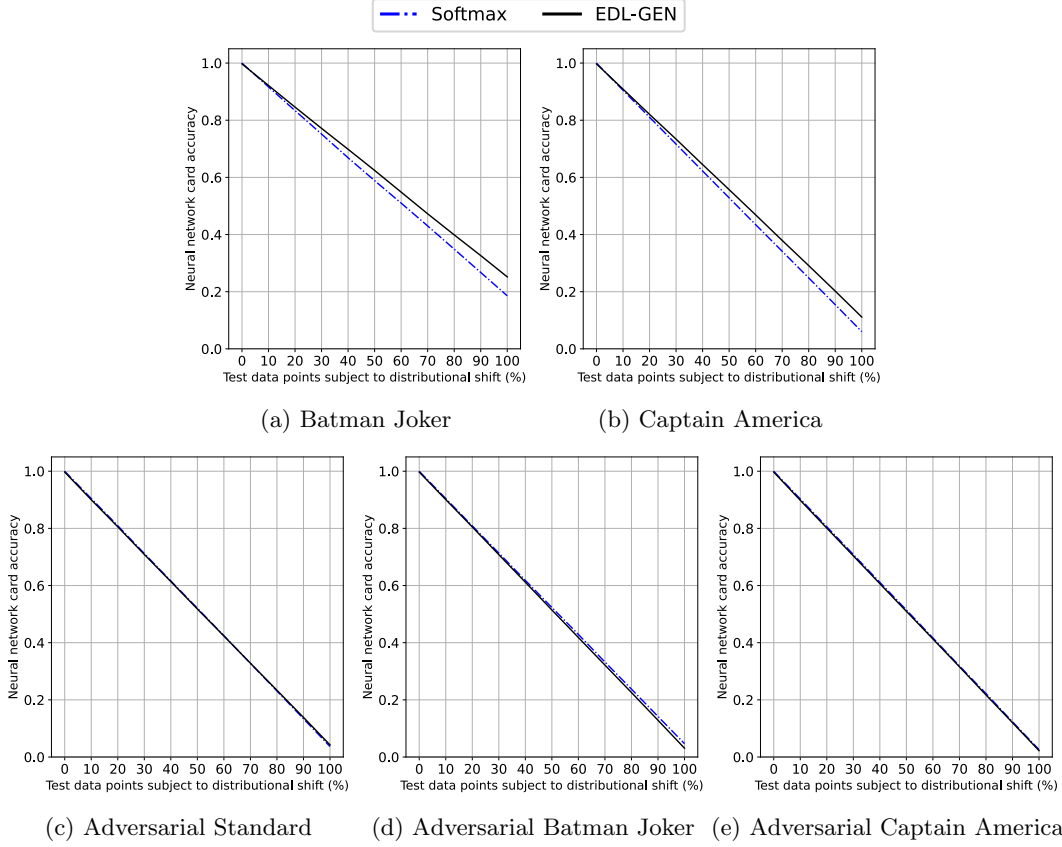


Figure 3.33: Neural network accuracy for predicting playing cards when test data points were subject to distributional shifts. Follow Suit Winner task.

the 9×9 task, FF-NSL outperforms all baselines, with the exception of the RF with additional background knowledge, which performed similarly to FF-NSL Softmax. It is indeed interesting to analyse why for the 9×9 task, FF-NSL Softmax outperforms FF-NSL EDL-GEN, especially in the cases where low percentages of test data are subject to distributional shift. Aside from the accuracy of the learned hypothesis, there are two contributing factors to the test set accuracy shown in Figure 3.34b; the ability to correctly predict test examples with and without distributional shift applied. In the 9×9 task, there are many more digit images on the grid for the neural network to predict. For test examples that are not subject to distributional shift, just one single incorrect neural network prediction may result in a miss-classified example. At 0% shift in Figure 3.34b, FF-NSL Softmax outperforms FF-NSL EDL-GEN. Now, the Softmax neural network accuracy over unseen and *non-rotated* MNIST digits is 0.9927, whereas for the EDL-GEN neural network it is 0.9861. This explains the drop in performance for FF-NSL EDL-GEN at 0% shift. As the shift is applied to the test set for percentages ranging between 10 – 80%, both FF-NSL Softmax and FF-NSL EDL-GEN fail to classify most examples subject to the shift, but FF-NSL EDL-GEN also fails to classify more examples that were not subject to the shift, when compared to FF-NSL Softmax. At shift $> 80\%$, the accuracy of the learned rules also becomes a factor, and the performance of both approaches deteriorates towards 50% accuracy.

3.5.3 Real-world Datasets

Figure 3.35 presents the accuracy of the entire FF-NSL framework, when evaluated over test data also subject to the same percentage of distributional shift as used during training. In the Crop

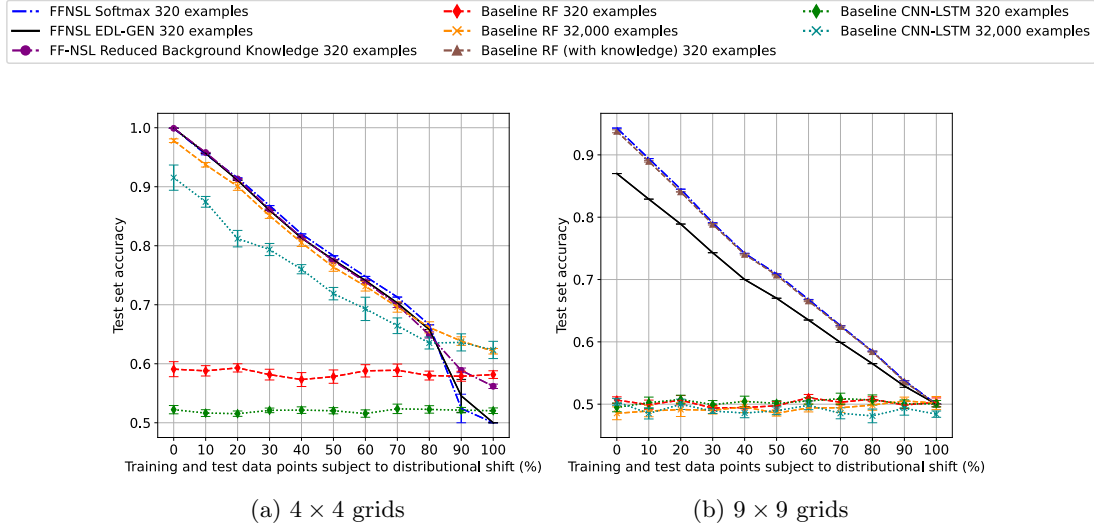


Figure 3.34: Accuracy of the FF-NSL framework when training and test data are subject to distributional shifts. Sudoku Grid Validity task.

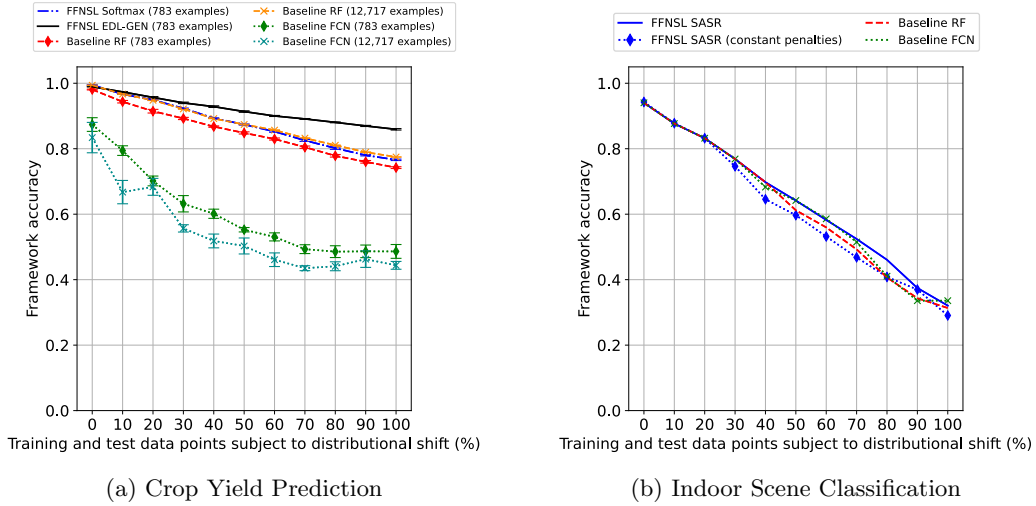
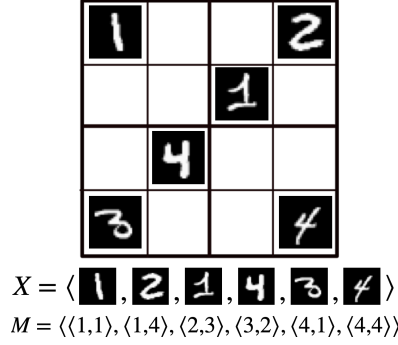


Figure 3.35: Accuracy of the FF-NSL framework when training and test data were subject to distributional shifts. Real-world datasets.

Yield Prediction task (Figure 3.35a), FF-NSL EDL-GEN outperforms all other methods, and FF-NSL Softmax outperforms all other methods trained with the same amount of data. The next best approach, the RF, requires significantly more data to match the performance of FF-NSL Softmax. The superior performance of FF-NSL EDL-GEN compared to FF-NSL Softmax is due to the EDL-GEN neural network predicting more accurately for images subject to distributional shift (see Figure 3.11). In the Indoor Scene Classification task, FF-NSL performs similarly to the best baseline approach, and all approaches degrade gracefully as the percentage of data points subject to distributional shift increases.

To summarise, our evaluation of FF-NSL on real-world datasets demonstrates the framework can support a wide range of neural network models, and solve problems closer to real-world applications. When taking into account the Follow Suit Winner and Sudoku Grid Validity results, we have also shown that FF-NSL can learn complex, first-order symbolic knowledge, using essential aspects of common-sense learning and reasoning, such as negation as failure and

Figure 3.36: Example data point from the Sudoku Grid Validity 4×4 task.

predicate invention. Also, we have shown the D2K component of FF-NSL is flexible enough to support a wide range of tasks, and integrate different neural and symbolic components.

3.6 RQ3: Evaluating the FF-NSL Framework with Uncertainty

Our final evaluation investigates whether FF-NSL can be configured to output uncertainty information during inference. This is particularly useful in the presence of distributional shifts, when the neural networks may predict the incorrect facts, and therefore, the downstream prediction may be incorrect. We configure FF-NSL to use the probabilistic programming language ProbLog (De Raedt et al., 2007), to generate downstream predictions with an associated probability. We encode the learned hypothesis, together with any background knowledge, into a ProbLog program. The neural network predictions are also encoded, using *annotated disjunctions*, which define a probability distribution over the choice of possible values for the concepts. The probability distributions are output by the neural network(s) during inference. Formally, for an image $\mathbf{x}_i \in X$, and associated meta-data $m_i \in M$, an annotated disjunction is defined as: $\eta_i(\mathbf{x}_i)[1] :: F(1, m_i) ; \dots ; \eta_i(\mathbf{x}_i)[k] :: F(k, m_i)$, where $\eta_i(\mathbf{x}_i)[k]$ defines the probability of prediction k , and $F(k, m)$ is a symbolic ground fact denoting k , and the associated meta-data, in a format relevant to the learned hypothesis of the given task. For example, consider the Sudoku Grid Validity 4×4 task, and the data point shown in Figure 3.36. If the neural network outputs the following probability distributions, for each $\mathbf{x}_i \in X$:

$$\begin{aligned}
 \eta_1(\mathbf{x}_1) &= [0.91, 0.01, 0.03, 0.05] \\
 \eta_2(\mathbf{x}_2) &= [0.02, 0.93, 0.01, 0.04] \\
 \eta_3(\mathbf{x}_3) &= [0.85, 0.08, 0.04, 0.03] \\
 \eta_4(\mathbf{x}_4) &= [0.08, 0.08, 0.09, 0.75] \\
 \eta_5(\mathbf{x}_5) &= [0.01, 0.01, 0.98, 0.00] \\
 \eta_6(\mathbf{x}_6) &= [0.02, 0.01, 0.01, 0.96]
 \end{aligned}$$

the corresponding annotated disjunctions can be expressed as:

```

0.91 :: value("1,1",1); 0.01 :: value("1,1",2); 0.03 :: value("1,1",3); 0.05 :: value("1,1",4).
0.02 :: value("1,4",1); 0.93 :: value("1,4",2); 0.01 :: value("1,4",3); 0.04 :: value("1,4",4).
0.85 :: value("2,3",1); 0.08 :: value("2,3",2); 0.04 :: value("2,3",3); 0.03 :: value("2,3",4).
0.08 :: value("3,2",1); 0.08 :: value("3,2",2); 0.09 :: value("3,2",3); 0.75 :: value("3,2",4).
0.01 :: value("4,1",1); 0.01 :: value("4,1",2); 0.98 :: value("4,1",3); 0.00 :: value("4,1",4).
0.02 :: value("4,4",1); 0.01 :: value("4,4",2); 0.01 :: value("4,4",3); 0.96 :: value("4,4",4).

```

Intuitively, this represents a choice over the possible neural network predictions, annotated with their respective probabilities. ProbLog is able to use the annotated disjunctions to calculate a complete probability distribution over the possible outputs. For example, given the correct rules of the Sudoku task, we can obtain a predicted probability for both *valid* and *invalid* outputs, using the following ProbLog program:

```

% <Annotated Disjunctions from above>
% <Sudoku Grid Validity Background Knowledge>

invalid :- value(V2,V3), value(V1,V3), block(V1,V0), block(V2,V0), V1 \= V2.
invalid :- value(V0,V2), col(V0,V3), col(V1,V3), value(V1,V2), V0 \= V1.
invalid :- row(V0,V3), value(V0,V2), value(V1,V2), row(V1,V3), V0 \= V1.

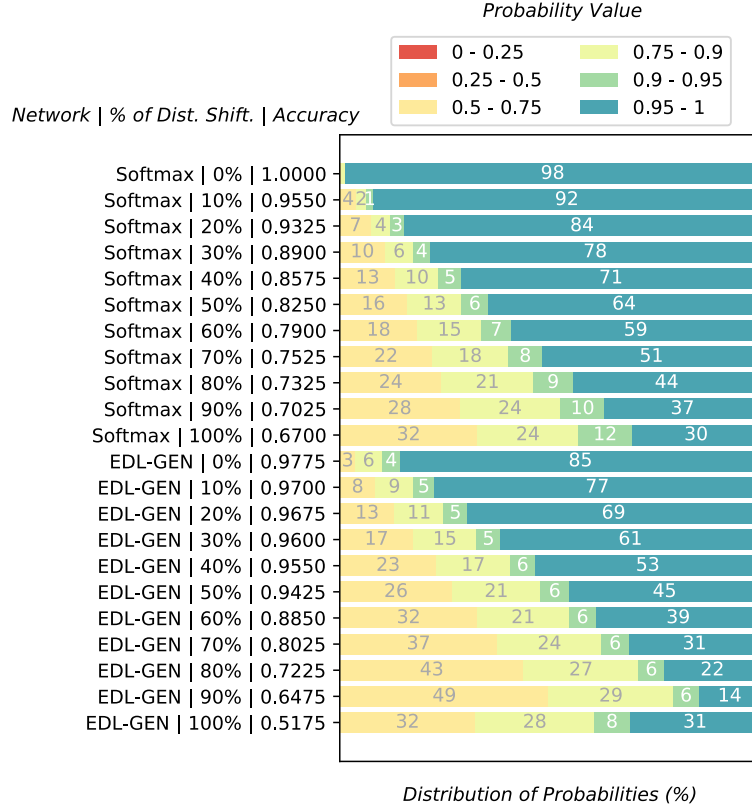
valid :- \+ invalid.
query(valid).
query(invalid).

```

where `\=` represents “not equal to”, the rule “`valid :- \+ invalid.`” states that the output must be *valid*, if *invalid* is *not* true, and the special “`query`” facts generate the output probabilities. For our example discussed thus far, the output probabilities are 1, and 0, for *valid*, and *invalid* respectively, given the high confidence of the neural network predictions.

To illustrate the potential of this approach, we evaluate the downstream predictive accuracy and confidence score distribution output by ProbLog, at each % of distributional shifts, on the Sudoku Grid Validity 4×4 task. The accuracy is calculated by taking the prediction with the maximum probability, and comparing it to the ground-truth label. The accuracy results presented here are not directly comparable to those in Figure 3.34a, as we run a focused evaluation on a smaller dataset, containing 80 test examples. The accuracy results are averaged over 5 repeats, using 5 different test sets, and the confidence score distributions are also plotted accounting for all predictions across the 5 test sets. The results are shown in Figure 3.37, for both variants of FF-NSL.

It is clear that the output probabilities are lower, as the percentage of distributional shift increases. For example, considering the case when the softmax neural network is used to extract concepts from the raw images, at 10% shift, 92% of the output probabilities are between 0.95 and 1. At 90% shift, only 37% of the output probabilities fall in this range. The EDL-GEN neural network provides a small advantage here, as the predicted probabilities are often lower than the corresponding probabilities when the softmax neural network is used. For example, at 90% shift, 49% of the probabilities are between 0.5 and 0.75, compared to 32% when softmax is used. However, our results highlight a challenge with using this method. For EDL-GEN, at 100% shift, the output probabilities often contain higher values than at 90% shift. On the surface, this appears to be counter-intuitive, how can the output be more certain, when the accuracy

Figure 3.37: FF-NSL Framework Uncertainty Analysis, Sudoku Grid Validity 4×4 task.

reduces? The problem is that the incorrect hypotheses are learned in this case. On one repeat, the learned hypothesis is `invalid : -value(V0,V1).`, which means, a Sudoku grid is invalid if there exists a value in any of the cells. In this case, since there are always values, the output probability is 1 (for `invalid`). Note the probability values in the annotated disjunction for each image sum to 1, so there is always a value defined, even if the neural network predicts with an even distribution. This method assumes that the hypotheses are correct, with probability 1.0 for each rule. If a probabilistic symbolic learner was used, one could annotate each rule in the hypothesis with a probability. In our experiments, given FF-NSL learns the correct rules when up to $\sim 80\%$ of the test examples are subject to distributional shift, this issue is mitigated in most cases.

3.7 Summary

In this chapter, we have introduced a new framework, called FF-NSL, which makes a first step towards integrating neural networks with symbolic learning, to enable interpretable rules to be learned from raw data. The novel aspect of our approach, the D2K generator, transforms neural network predictions to symbolic facts, and also aggregates neural network confidence scores to create a weight for the symbolic learner. This provides an informative bias, such that the symbolic learner is able to learn a hypothesis that covers examples with correct neural network predictions. In our comprehensive experimental analysis, we have demonstrated the broad applicability and effectiveness of this framework, in terms of the accuracy of the learned hypotheses, and their interpretability. We have also explored in-depth the effect of using an uncertainty-aware neural network, based upon evidential deep learning, when calculating the

example weight penalties. We have demonstrated an advantage over using a standard Softmax-based neural network, when training data points are subject to distributional shifts. This is the first study to investigate symbolic learning from raw data under these conditions, and our results indicate the LAS symbolic learners, ILASP and FastLAS, are highly tolerant to incorrect neural network predictions, in the presence of distributional shifts in the input data. The results of this chapter are encouraging for others in the field, and in industry, seeking to use symbolic learning from raw inputs.

However, there is one major drawback, in that FF-NSL assumes the perception component is pre-trained, using a large labelled dataset. This may not be a realistic assumption in practice, as such a dataset may not be available, or, may be expensive to obtain. In the next chapter, we relax this assumption, and develop an architecture capable of training both neural and symbolic components in an end-to-end fashion.

Chapter 4

Iterative Neuro-Symbolic Learning

To articulate the goals of this chapter, let us first consider the field of deep learning. One of the reasons deep learning has grown significantly in popularity, is due to the flexible nature of the learning system. It is possible to construct a high-performance learning architecture to solve a given task, by composing, and combining, a wide variety of neural networks. Learning is divided into two steps: (1) A *forward pass* through the neural network architecture, that transforms the raw input data into the desired output representation, and (2) a *backward pass*, which updates the parameters of the neural network via gradient descent. The gradients are calculated based upon a loss function, that computes the error in the neural network prediction(s) compared to (a) ground-truth label(s). This is known as training the neural network *end-to-end*, as the computation continuously moves forwards and backwards many times, mapping the input training data to a candidate output representation, and then updating the network parameters based on the error signal from the loss function. The goal is to improve the performance on subsequent iterations, until the loss is minimised. Aside from engineering the architecture of the neural network, and tuning the hyper-parameters, the user only has to provide a dataset of inputs and outputs, and the neural network is able to learn everything else, i.e., the function that maps the inputs to the outputs. Whilst there are many reasons to celebrate the success of deep learning, there are also significant drawbacks, that limit its adoption in industry, and also pose a real challenge for AI safety. For example, the lack of interpretability, and out-of-distribution generalisation. Therefore, there is wide interest from the neuro-symbolic community in building a flexible *end-to-end* learning framework, that combines the benefits of deep learning, with the robustness of symbolic AI (Garcez et al., 2022). A popular direction is to develop an integrated architecture that can accept raw data as input using neural computation, before reasoning over, or learning, explicit symbolic knowledge that is interpretable and robust to new tasks outside the training distribution. However, the challenge in such an integration, is that pure symbolic components are not differentiable, and therefore, it becomes difficult to directly integrate symbolic computation with neural networks in an end-to-end fashion, as it is not possible to back-propagate gradients through a non-differentiable symbolic component.

Nevertheless, various approaches have tackled this problem via a technique called *neuro-symbolic reasoning*. These approaches demonstrate it is possible to train a neural network to extract primitive concepts from raw data, by abducting the set of relevant concepts required to infer a label of a downstream task, with respect to a complete background knowledge. This

effectively bridges the gap between the differentiable (neural) and non-differentiable (symbolic) computation, as the abduced concepts can be used to train the neural component. To avoid engineering the knowledge manually, it is interesting to consider whether we can replace the symbolic knowledge in these neuro-symbolic reasoning approaches, with a symbolic learner, and thus learn both neural and symbolic components to simulate an end-to-end training approach.

In this chapter, we introduce a new framework, called NSIL, that achieves *iterative* neuro-symbolic learning to train both neural and symbolic components. NSIL first learns a candidate hypothesis during an initial bootstrap phase, which is then used to perform a round of neural network training using neuro-symbolic reasoning. A set of corrective examples are automatically created for the symbolic learner, based on the current training set performance, to bias towards either retaining, or refining, the current hypothesis. Subsequent iterations are performed, whereby the neural network is trained using a fixed hypothesis, and then the hypothesis is updated (if necessary) for the next iteration. Our experiments demonstrate NSIL is able to learn the correct knowledge, and train the neural network to classify concepts from raw data, on a variety of tasks. Note that the key advantage of NSIL is that no label information is required for the concept values, only the possible space of concept values is assumed. This simulates an end-to-end training paradigm, where the only data required is the raw inputs, and the final outputs. Compared to other approaches, NSIL is the first neuro-symbolic framework capable of learning complex and expressive knowledge in the language of ASP, whilst training a general neural network to extract concepts from raw input data.

The remainder of this chapter is organised as follows. In Section 4.1, we formalise the problem statement of training both neural and symbolic components. In Section 4.2, we introduce our NSIL framework, and present our novel contribution of biasing the symbolic learner using corrective examples. We evaluate NSIL in Section 4.3, using three problem domains that require learning complex, and expressive knowledge, including an NP-complete problem. We demonstrate NSIL achieves state-of-the-art performance, compared to a variety of neural baselines, and where appropriate, *MetaAbd*, the closest existing neuro-symbolic baseline (Dai and Muggleton, 2021). We outline a set of challenges with scaling NSIL to more complex problems in Section 4.4, and, finally, summarise in Section 4.5. Some of the work in this chapter was presented as a long talk at the 2022 AAAI Fall Symposium “*Thinking Fast and Slow and Other Cognitive Theories in AI*”, and was subsequently published at the 2023 International Joint Conference on Artificial Intelligence (Cunnington et al., 2023b).

4.1 Problem Formulation

Our problem statement largely follows Chapter 3. We again consider tasks where the input is a sequence of raw data in the form of images, and the output represents a higher-level prediction that requires reasoning over the primitive concepts observed in the input. As in Chapter 3, the knowledge needs to be learned, but now the primitive concepts are latent, since no label information is assumed for the values of these concepts. We again decompose the task into a perception function $g : \mathcal{X} \rightarrow \mathcal{Z}$, and a reasoning function $h : (\mathcal{Z} \times \mathcal{M})^n \rightarrow \mathcal{Y}$. g maps a raw image $\mathbf{x} \in \mathcal{X}$ into a primitive concept $z \in \mathcal{Z}$, and h maps n pairs of concepts and associated meta-data $m \in \mathcal{M}$ to a target label $y \in \mathcal{Y}$. In this chapter, we assume h accepts a variable-length sequence of concept meta-data pairs as input, where n is $a \leq n \leq b$. The reasoning function is defined as $h = H \cup B$, where H is a *hypothesis* that performs the required reasoning, and B is an optional background knowledge that defines a set of predicates and facts relevant to the task. B is

assumed to be given, and H needs to be learned. Consider a training set $\mathcal{D}_{\text{task}} = \{\langle X, M, y \rangle, \dots\}$, where $X = \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$ is an n -length sequence of images, $M = \langle m_1, \dots, m_n \rangle$ is an n -length sequence of meta-data associated with X , and $y \in \mathcal{Y}$ is a target label. The goal is to learn g and h , such that:

$$h(\langle (g(\mathbf{x}_1), m_1), \dots, (g(\mathbf{x}_n), m_n) \rangle) = y \text{ for all } \langle X, M, y \rangle \in \mathcal{D}_{\text{task}} \quad (4.1)$$

Intuitively, this means to output correct predictions for all of the training data points, where h outputs the correct target label, given a sequence of pairs of concept predictions from g , and the meta-data. In this chapter, to simplify notation, we use $Z = \langle g(\mathbf{x}_1), \dots, g(\mathbf{x}_n) \rangle$ to denote the sequence of concept value predictions from the perception function, where $z_i = g(\mathbf{x}_i)$. The goal is to learn g and h *jointly*, without any supervision for the concepts. This is challenging, as h is not differentiable, and cannot be learned using standard gradient-based optimisation. To assist the reader, we introduce a running example that explains the NSIL framework when applied to the addition variant of our Two-Digit Arithmetic task.

Example 4.1.1. *This task is based upon the MNIST Addition task from (Manhaeve et al., 2018). A training data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ contains a pair of MNIST images (Deng, 2012), labelled with the sum of the digit values. For example, let $X = \langle \text{2}, \text{1} \rangle$, and $y = 3$. In this example, the meta-data is empty i.e., $M = \langle \rangle$. Once trained, g extracts a digit concept for each image in X , and h maps the sequence of digit concepts, and the meta-data, to the downstream label (i.e., it performs the addition of the two digits). In this example, the goal is to learn g and h s.t. $g(\mathbf{x}_1) = 2$, $g(\mathbf{x}_2) = 1$, and $h(\langle (2, -), (1, -) \rangle) = 3$, where $-$ is used to denote no meta-data.*

Let us now introduce the NSIL framework, and explain how g and h are learned.

4.2 NSIL Framework

The NSIL framework relies upon three major components: (1) A **symbolic learner**, which learns a hypothesis. (2) A **neural network**, which extracts primitive concepts from the raw input data. (3) **Corrective examples**, to bias the symbolic learner to either retain or refine the hypothesis, depending on the overall system performance, and the neural network confidence when extracting concept values. An overview of the NSIL framework is presented in Figure 4.1. During learning, NSIL trains a neural network using a candidate hypothesis learned by a symbolic learner. Corrective examples then bias the symbolic learner to retain or refine the hypothesis for the next iteration. During inference, the trained neural network forms the perception function g , and the learned hypothesis forms the reasoning function h , to infer predictions $\hat{y} \in \mathcal{Y}$ from unseen raw data. Let us now describe the optimisation of the neural and symbolic components.

4.2.1 Symbolic Component

The symbolic component is a LAS symbolic learner. Recall from Chapter 2 the input is a domain knowledge B that contains any (optional) background knowledge, and a search space $\mathcal{H}$. The symbolic learner also accepts a set of WCDPI examples E as input, where each example associates a sequence of concept values Z , and associated meta-data M , with a target label y . The output is a learned hypothesis $H \in \mathcal{H}$ which is a first-order ASP program. H , together with B , forms the reasoning function $h = H \cup B$ that maps sequences of concept values, and associated meta-data, to target labels. In this chapter, we configure each example $e \in E$ to represent the label either positively (i.e., y *should* be output for Z and M), or negatively (y should *not* be

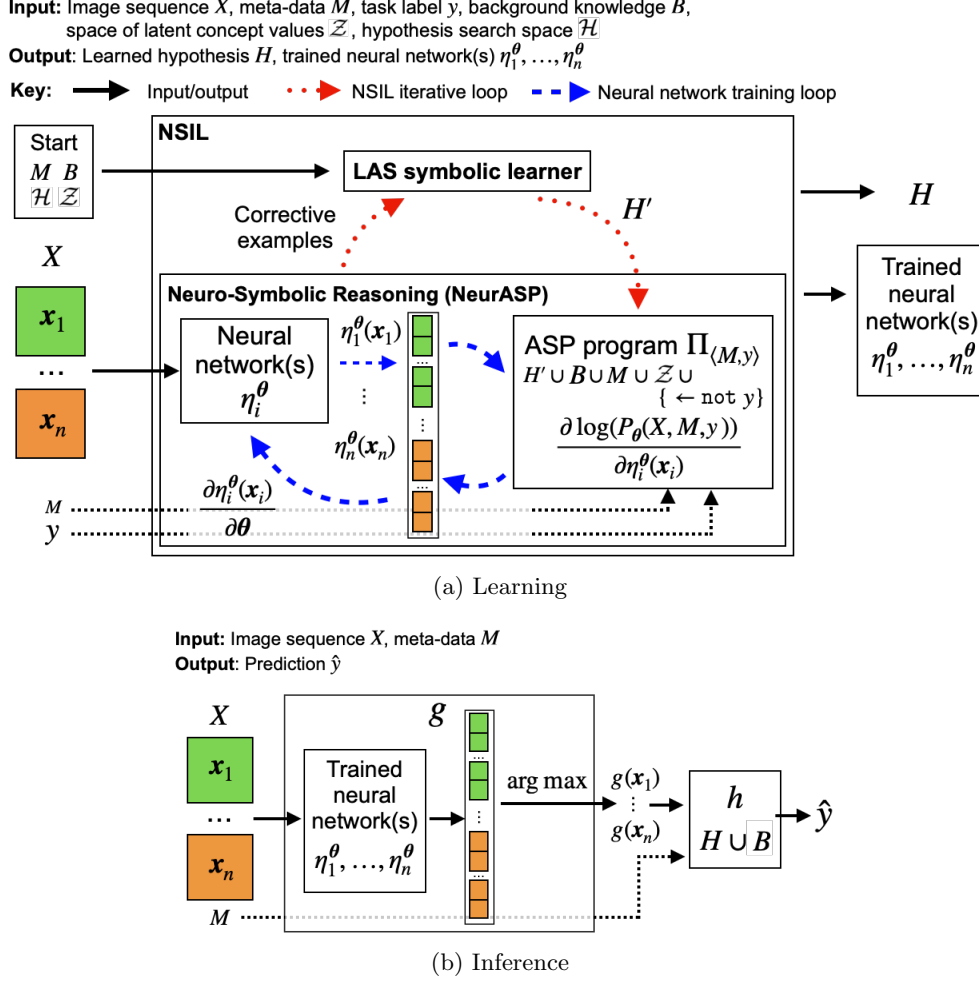


Figure 4.1: (a) NSIL learning with a single data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. (b) NSIL inference with a single input $\langle X, M \rangle$.

output for Z and M). H is said to *cover* an example if $H \cup B$ outputs the target label, for a given Z and M . Details on how the examples are constructed are presented in Section 4.2.3. During NSIL learning, we add constraints to B , to ensure H is guaranteed to output each target label we observe in $\mathcal{D}_{\text{task}}$, i.e., for each $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$, there must exist a sequence of concept values $Z \subseteq \mathcal{Z}$ s.t. $h(\langle (z_1, m_1), \dots, (z_n, m_n) \rangle) = y$. As presented in Chapter 3, the learned H minimises a scoring function based on the length of H and the coverage over the example set E . The LAS symbolic learner solves the optimisation given in Equation 2.27.

4.2.2 Neural Component

The perception function g contains a sequence of neural networks $\langle \eta_1^\theta, \dots, \eta_n^\theta \rangle$, where each η_i^θ extracts a primitive concept from an image $x_i \in X$. θ denotes the neural network parameters which need to be learned. Recall that X is the sequence of images observed in a training data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. Depending on the given task, there could be many configurations of the sequence of neural networks. For example, the same neural network may be able to extract the same concept from all images in X , or, each image in X may require a different neural network to extract a different concept. There could also be many combinations thereof, where the same neural network applies to a subsequence of X . Without loss of generality, we denote η_i as the

neural network capable of extracting a primitive concept from the image $\mathbf{x}_i \in X$. Formally, let us denote a neural network as the function $\eta : \mathcal{X} \rightarrow [0, 1]^k$ which maps a raw image to a k -length vector denoting a probability distribution over k concept values. We define the neural network prediction for an image $\mathbf{x} \in X$, as the index in the output vector with the largest probability:

$$g(\mathbf{x}) = \arg \max_{j \in \{1, \dots, k\}} (\eta^\theta(\mathbf{x})[j]) \quad (4.2)$$

In this chapter, we are concerned with training the neural network when no concept labels are available. Therefore, we adopt a neuro-symbolic reasoning approach, and, for each data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$, we train the neural network using a learned H , $\mathcal{Z}$, B , and y . Specifically, as H is in the language of ASP, we use the neuro-symbolic reasoning system called NeurASP (Yang et al., 2020), which optimises a semantic loss function (Xu et al., 2018) for neural network training. Informally, for each data point, the neural network is trained to predict the concept values that, together with the meta-data and $H \cup B$ infer y .

Example 4.2.1. *Following Example 4.1.1, let us assume H is the correct addition hypothesis. For $X = \langle \text{2}, \text{7} \rangle$, $M = \langle \rangle$, and $y = 3$, the neural network is trained to output either $\langle 0, 3 \rangle$, $\langle 1, 2 \rangle$, $\langle 2, 1 \rangle$, or $\langle 3, 0 \rangle$ for these images, as these concept values all yield the output $y = 3$, given $H \cup B$.*

Formally, consider a data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$. We define an ASP program $\Pi_{\langle M, y \rangle} = H \cup B \cup M \cup \mathcal{Z} \cup \{\leftarrow \text{not } y\}$, with answer sets denoted $AS(\Pi_{\langle M, y \rangle})$. Each answer set contains a sequence of concept values Z that output y , given H and B , i.e., assuming the reasoning function $h = H \cup B$, $h(\langle (z_1, m_1), \dots, (z_n, m_n) \rangle) = y$, where $z_1, \dots, z_n \in Z$. We define the probability of each $Z \in AS(\Pi_{\langle M, y \rangle})$ in terms of the neural network confidence when predicting each concept value:

$$P_\theta(X, Z) = \prod_{z_i \in Z, \mathbf{x}_i \in X} \eta_i^\theta(\mathbf{x}_i)[z_i] \quad (4.3)$$

The probability of an $\langle X, M, y \rangle$ data point is then defined as:

$$P_\theta(X, M, y) = \sum_{Z \in AS(\Pi_{\langle M, y \rangle})} P_\theta(X, Z) \quad (4.4)$$

and the neural network is trained to optimise:

$$\theta^* = \arg \max_{\theta} \left[\sum_{\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}} \log(P_\theta(X, M, y)) \right] \quad (4.5)$$

Intuitively, this can be interpreted as training the neural network to maximise the probability of the concept values that output a downstream label, given H and B . Since there are no concept labels to use within a loss function, NeurASP calculates the gradients of each neural network output directly, by calculating:

$$\frac{\partial \log(P_\theta(X, M, y))}{\partial \eta_i^\theta(\mathbf{x}_i)} \quad (4.6)$$

Example 4.2.2. *Following Example 4.2.1, let us assume the neural network outputs the following probability distributions. $k = 10$, as there are 10 possible MNIST digit values 0–9.*

$$\begin{aligned}\eta_1^\theta(\mathbf{x}_1) &= [0.02, 0.01, 0.87, 0.01, 0.01, 0.01, 0.02, 0.01, 0.02, 0.02] \\ \eta_2^\theta(\mathbf{x}_2) &= [0.01, 0.75, 0.03, 0.01, 0.01, 0.01, 0.02, 0.14, 0.01, 0.01]\end{aligned}$$

There are four possible answer sets of the ASP program for this example. The answer sets include:

$$AS(\Pi_{\langle M, y \rangle}) = \{\langle 0, 3 \rangle, \langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 3, 0 \rangle\}$$

which represent the possible concept values that satisfy the downstream label $y = 3$ (assuming the correct H). The probability of each sequence of concept values $Z \in AS(\Pi_{\langle M, y \rangle})$ can then be calculated as:

$$\begin{aligned}P_\theta(X, \langle 0, 3 \rangle) &= 0.02 \times 0.01 = 0.0002 \\ P_\theta(X, \langle 1, 2 \rangle) &= 0.01 \times 0.03 = 0.0003 \\ P_\theta(X, \langle 2, 1 \rangle) &= 0.87 \times 0.75 = 0.6525 \\ P_\theta(X, \langle 3, 0 \rangle) &= 0.01 \times 0.01 = 0.0001\end{aligned}$$

The probability of the $P_\theta(\langle \text{2}, \text{7} \rangle, \langle \rangle, 3)$ data point is therefore:

$$P_\theta(\langle \text{2}, \text{7} \rangle, \langle \rangle, 3) = 0.0002 + 0.0003 + 0.6525 + 0.0001 = 0.6531 \quad (4.7)$$

The neural network is then trained to maximise the probability of all data points in the training set via gradient descent. Note that each of these concept values receives a gradient update, and in this example, the network learns to distinguish individual digits by observing multiple data points.

Now we have outlined the neural and symbolic components, we can introduce our novel method of integration. Recall that the symbolic learner learns a hypothesis which is used to train the neural network. An immediate question is, what happens if the learned hypothesis is incorrect, as training with an incorrect hypothesis may lead to a sub-optimal neural network? To tackle this problem, NSIL adopts an iterative training procedure where both the symbolic learner, and the neural network, are updated on each iteration. This integration relies on a set of weighted *corrective examples* for the symbolic learner, that encourages the current hypothesis to be retained, or refined on the next iteration by exploring the hypothesis space. Let us now outline how the corrective examples are structured and how their weights are updated on each iteration.

4.2.3 Corrective Examples

A corrective example $e_{Z, M, y} \in E$ associates a sequence of concept values Z , and related meta-data M , with a target label y . Each example is configured to represent the label either positively (i.e., y should be output for a given Z and M , denoted $e_{Z, M, y}^{\text{pos}}$), or negatively (y should *not* be output for a given Z and M , denoted $e_{Z, M, y}^{\text{neg}}$). Also, recall that each example is weighted with a penalty, paid if the example is left *uncovered* by the learned hypothesis. A pair of corrective examples $\langle e_{Z, M, y}^{\text{pos}}, e_{Z, M, y}^{\text{neg}} \rangle$ represents both positive and negative cases for a given $\langle Z, M, y \rangle$ tuple.

The set of symbolic examples E contains a pair of corrective examples for all possible $\langle Z, M, y \rangle$ tuples, and initially, all examples have equal weight set to 0. This means there is no bias given to the symbolic learner as to which examples should be covered. We restrict the weights to the interval $[0, 100]$, and the goal is to maximise the weights of the positive examples that contain correct combinations of concept values with target labels, whilst minimising the weights of the corresponding negative examples.

On each NSIL iteration, the neural network is trained with a candidate hypothesis, denoted H' . The corrective example weights are then updated, using two sources of information: (1) The overall performance when predicting training set labels y . (2) The neural network confidence when predicting the sequence of concepts Z . We calculate a weight value, and associate this with a specific $\langle Z, M, y \rangle$ tuple. To capture the overall performance, we obtain $\hat{y}$ predictions with a forward pass over the training set, using both neural and symbolic components. We then calculate the False Negative Rate (FNR) of each unique meta-data and label combination, which depends on the current H' and current neural network parameters, denoted θ' :

$$\text{FNR}_{H', \theta'}(M, y) = 100 \times \left(1 - \frac{TP_{H', \theta'}(M, y)}{TP_{H', \theta'}(M, y) + FN_{H', \theta'}(M, y)} \right) \quad (4.8)$$

where $TP_{H', \theta'}(M, y)$ and $FN_{H', \theta'}(M, y)$ are the number of true positives and false negatives of training data points with meta-data M and label y , respectively. We multiply by 100 to map the FNR into our $[0, 100]$ weight range. Intuitively, if NSIL accurately predicts training data points with meta-data M , and label y , $\text{FNR}_{H', \theta'}(M, y)$ will be close to 0, otherwise $\text{FNR}_{H', \theta'}(M, y)$ will be close to 100. To associate the FNR with a $\langle Z, M, y \rangle$ tuple, we obtain all possible Z with meta-data M , that currently output label y given the current H' , by intercepting the answer sets of $\Pi_{\langle M, y \rangle}$ within NeurASP. We divide the FNR by the number of answer sets of $\Pi_{\langle M, y \rangle}$, i.e.:

$$\frac{\text{FNR}_{H', \theta'}(M, y)}{|AS(\Pi_{\langle M, y \rangle})|}$$

as there could be multiple Z with a certain M , that lead to a given y . For instance, in Example 4.2.1, there are four sequences of concept values that lead to the label $y = 3$, so the FNR is shared between these possibilities.

To capture the neural network confidence, and associate it with a $\langle Z, M, y \rangle$ tuple, we compute an average confidence value over all training data points with meta-data M , and label y , where the neural network predicts Z . Formally, let $D_{\langle Z, M, y \rangle}$ be a sequence of data points $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ with meta-data M , and label y , where the neural network predicts the sequence of concept values Z for the input sequence X , i.e., $\langle g(\mathbf{x}_1), \dots, g(\mathbf{x}_n) \rangle = Z$. Let us also define the probability of X as the product of the maximal neural network confidence score for each image in the sequence:

$$P_{\theta'}(X) = \prod_{\mathbf{x}_i \in X} \max(\eta_i^{\theta'}(\mathbf{x}_i)) \quad (4.9)$$

The overall confidence for a $\langle Z, M, y \rangle$ tuple is then defined as:

$$\text{CONF}_{\theta'}(Z, M, y) = \frac{100}{|D_{\langle Z, M, y \rangle}|} \times \sum_{\langle X, M, y \rangle \in D_{\langle Z, M, y \rangle}} P_{\theta'}(X) \quad (4.10)$$

Again, we multiply by 100 to map the average confidence score into our $[0, 100]$ weight range. Given our two sources of information, we now outline how the corrective example weights are updated. On each iteration, we first set the weights of the $\langle Z, M, y \rangle$ tuples that we have observed

from the answer sets of $\Pi_{\langle M, y \rangle}$ as follows. For clarity, we use the notation $W(\cdot)$ to denote the example weight penalty e_{pen} of each example within the corrective example pair.

$$W(e_{Z, M, y}^{\text{pos}}) = 0 \quad (4.11)$$

$$W(e_{Z, M, y}^{\text{neg}}) = \frac{\text{FNR}_{H', \theta'}(M, y)}{|\text{AS}(\Pi_{\langle M, y \rangle})|} \quad (4.12)$$

The negative example weight is set by the FNR. If the FNR is large, this will encourage a different label for Z . The positive example weight is set to 0 because the FNR may apply to multiple $\langle Z, M, y \rangle$ tuples, and may not indicate any positive information for an individual tuple. We then update the example weights using the neural network confidence to ensure positive information for an individual $\langle Z, M, y \rangle$ tuple is captured:

$$W(e_{Z, M, y}^{\text{pos}}) = W(e_{Z, M, y}^{\text{pos}}) + \lambda \text{CONF}_{\theta'}(Z, M, y) \quad (4.13)$$

$$W(e_{Z, M, y}^{\text{neg}}) = W(e_{Z, M, y}^{\text{neg}}) - \lambda \text{CONF}_{\theta'}(Z, M, y) \quad (4.14)$$

where $\lambda \in [0, 1]$ controls the effect of the $\text{CONF}_{\theta'}(Z, M, y)$ update, and is set to 1 in all of our experiments (see Section 4.3.5 for a study investigating different values of λ). Example weights are allowed to persist across iterations, and in Equations 4.13 and 4.14, the W terms in the equation body could equal the weights set by Equations 4.11 and 4.12, or the weights set on a previous iteration, if a $\langle Z, M, y \rangle$ tuple predicted by the neural network does not exist in the answer sets of $\Pi_{\langle M, y \rangle}$. If there are no concept predictions of a certain Z , $\text{CONF}_{\theta'}(Z, M, y) = 0$. Finally, the weights are converted to an integer, and clipped to the range $[0, 100]$ to prevent a particular corrective example combination dominating the optimisation of the learned hypothesis. We chose this range since it matches the range of both the FNR, and the neural network confidence score, as well as being easy to interpret.¹ A pseudo-code listing of the corrective example weight updates is presented in Algorithm 1.

Example 4.2.3. *Following Example 4.2.1, let H' be the correct addition hypothesis, $y = 3$ and $\text{FNR}_{H', \theta'}(M, y) = 5$, indicating 95% of training data points with label 3 have been predicted correctly. In this task, since the meta-data is empty, the FNR is essentially computed per-label. Recall the answer sets of $\Pi_{\langle M, y \rangle}$ are $\{\langle 0, 3 \rangle, \langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 3, 0 \rangle\}$. Let us also assume that for training data points with label 3, the correct Z are predicted with 98% confidence, and $\lambda = 1$. Therefore, the positive corrective examples associated with these concept values all have weight 98, and the corresponding negative examples all have weight $(\frac{5}{4}) - 98 = -96.75$, which is clipped to 0. This encourages the symbolic learner to cover the positive examples on the next iteration, and leave the negative examples uncovered.*

Example 4.2.4. *Conversely, let H' be an incorrect hypothesis, $y = 3$, $\lambda = 1$, and $\text{FNR}_{H', \theta'}(M, y) = 95$, indicating only 5% of training data points with label 3 have been predicted correctly. Let us assume the answer sets of $\Pi_{\langle M, y \rangle}$ are $\{\langle 0, 4 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle\}$, and, for all data points labelled 3, the neural network only predicts $\langle 1, 2 \rangle$ with 75% confidence. Therefore:*

$$W(e_{\langle 1, 2 \rangle, 3}^{\text{pos}}) = 75, \quad W(e_{\langle 0, 4 \rangle, 3}^{\text{pos}}) = W(e_{\langle 2, 3 \rangle, 3}^{\text{pos}}) = 0,$$

¹Since we instantiate NSIL with ILASP (Law et al., 2020b) and FastLAS (Law et al., 2020a), in our experiments we shift the weights to the range $[1, 101]$, as these systems require integer weights > 0 . The weight range is kept fixed for all experiments.

Algorithm 1 NSIL Corrective Example Weight Updates

Input: Candidate hypothesis H' , neural network parameters θ' , and examples E from the previous iteration. On the first iteration, E contains all possible corrective examples with zero weight.

▷ 1) Analyse the overall performance.

▷ Get the unique combinations of meta-data sequences and labels in the training data.

$\text{UML} \leftarrow (\{\langle M, y \rangle : \langle X, M, y \rangle \in \mathcal{D}_{\text{task}}\})$

▷ For each combination.

for $\langle M, y \rangle \in \text{UML}$ **do**

▷ Get the answer sets of the ASP program.

$A \leftarrow AS(\Pi_{\langle M, y \rangle})$

▷ Calculate the false negative rate as a measure of the overall performance.

$F \leftarrow \text{FNR}_{H', \theta'}(M, y)$

▷ Normalise the false negative rate by the number of answer sets.

$\text{NF} \leftarrow \frac{\text{FNR}_{H', \theta'}(M, y)}{|A|}$

▷ Update the example weights for each sequence of concepts in the answer sets of the ASP program.

for $Z \in A$ **do**

▷ The weights of these examples are updated in-place within E .

$W(e_{Z, M, y}^{\text{pos}}) = 0$

$W(e_{Z, M, y}^{\text{neg}}) = \text{NF}$

▷ 2) Analyse the neural network confidence.

for $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ **do**

▷ Get the neural network predictions for this data point.

$Z = \langle g_{\theta'}(\mathbf{x}_1), \dots, g_{\theta'}(\mathbf{x}_n) \rangle$

▷ Calculate the aggregated confidence score for these predictions, meta-data, and label.

$C \leftarrow \text{CONF}_{\theta'}(Z, M, y)$

▷ Update the example weights associated with these predictions, meta-data, and label.

▷ The weights of these examples are updated in-place within E .

$W(e_{Z, M, y}^{\text{pos}}) = W(e_{Z, M, y}^{\text{pos}}) + \lambda C$

$W(e_{Z, M, y}^{\text{neg}}) = W(e_{Z, M, y}^{\text{neg}}) - \lambda C$

▷ 3) Ensure the weights are integers, and in the desired interval.

for $e \in E$ **do**

$W(e) = \text{CLIP}(\text{int}(W(e)), 0, 100)$

Return E

$$W(e_{\langle 0,4 \rangle,3}^{neg}) = W(e_{\langle 2,3 \rangle,3}^{neg}) = \frac{95}{3} = 31.67,$$

$$W(e_{\langle 1,2 \rangle,3}^{neg}) = \left(\frac{95}{3}\right) - 75 = -43.33 \text{ (which is clipped to 0)}.$$

This encourages the symbolic learner to cover $e_{\langle 1,2 \rangle,3}^{pos}$, $e_{\langle 0,4 \rangle,3}^{neg}$ and $e_{\langle 2,3 \rangle,3}^{neg}$ on the next iteration, and leave $e_{\langle 1,2 \rangle,3}^{neg}$, $e_{\langle 0,4 \rangle,3}^{pos}$, and $e_{\langle 2,3 \rangle,3}^{pos}$ uncovered.

In our evaluation in Section 4.3, we present instances of the corrective examples for each learning task considered. For computational efficiency, corrective example pairs with equal weights are not given to the symbolic learner, since they do not bias the symbolic learner in either direction. To summarise, NSIL solves Equations 2.27 and 4.5 with the following steps:

1. The symbolic learner learns an initial hypothesis H' that satisfies B , and covers each possible combination of meta-data $m \in \mathcal{M}$ and target label $y \in \mathcal{Y}$. This is guaranteed during the learning task construction. The neural network parameters θ' are initialised randomly.
2. The neural network is trained for 1 epoch to update θ' , using H' , within the ASP program $\Pi_{\langle M,y \rangle}$, for each $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$.
3. The weights of the corrective examples are updated (see Section 4.2.3 and Algorithm 1), and a new H' is learned.
4. Steps 2–3 are repeated for a fixed number of iterations. NSIL outputs the trained neural network(s) $\langle \eta_1^\theta, \dots, \eta_n^\theta \rangle$, and the learned hypothesis H .

4.3 Evaluation

Our evaluation addresses the following questions: (1) Can NSIL train both neural and symbolic components, learning general, expressive and interpretable knowledge from raw data? (2) Is NSIL vulnerable to becoming stuck in local optima when training with limited data? (3) Can NSIL learn the correct knowledge in large search spaces? (4) How does NSIL compare to the closest existing systems, including purely differentiable models? Finally, (5) can NSIL learn expressive solutions to complex tasks that would be difficult to learn and represent with other systems?

To answer these questions, we consider three problem domains: Cumulative Arithmetic, Two-Digit Arithmetic, and the Hitting Set problem. The Cumulative Arithmetic tasks follow (Dai and Muggleton, 2021) and enable a comparison to the closest existing system $Meta_{Abd}$, where NSIL achieves state-of-the-art performance. $Meta_{Abd}$ performs a joint abduction and induction process, where pseudo-labels for training the neural network are abducted at the same time the knowledge is learned. To concretely demonstrate the contribution of NSIL, the other domains require learning more expressive knowledge that $Meta_{Abd}$ cannot learn, including negation as failure, choice rules, and programs with multiple answer sets. We demonstrate NSIL trains both neural and symbolic components successfully, outperforms fully neural baselines, and learns the correct knowledge in large search spaces. This addresses questions 1, 3, and 4. Question 2 is also addressed here, where a reduced amount of training data is used in the Two-Digit Arithmetic domain, demonstrating that NSIL is able to train successfully under these conditions. Question 5 is addressed using the Hitting Set domain, demonstrating that NSIL is able to solve an NP-complete problem and generate *all* the hitting sets of a given collection, despite learning from binary labels that only indicate the *existence* of a hitting set.

4.3.1 Experimental Setup

Baselines. Aside from the comparison to *MetaAbd*, we compare to Concept Bottleneck Models (CBMs) (Koh et al., 2020), which are end-to-end neural networks that model both perception and reasoning components. We ensure the perception network is the same as in NSIL, but the reasoning network is a MLP instead of a symbolic learner. We train two variants end-to-end, using the same weak supervision as NSIL (i.e., no concept labels are used). One variant (denoted *CBM*) uses a linear layer as output from the perception network, and the second variant (denoted *CBM-S*) uses a softmax layer, replicating more closely the perception network used in NSIL. These CBMs are chosen to reflect a fully neural architecture that performs perception and reasoning, in a similar fashion to NSIL.

For the Two-Digit Arithmetic tasks, we also evaluate: (1) The CNN from (Manhaeve et al., 2018) that accepts a concatenated set of images as input, and is trained to output the final prediction directly. This architecture is largely the same as the NSIL CNN, except the size of the output layer is 19, the network accepts a pair of concatenated images as input, and a log softmax layer is used to provide a classification. This is chosen to highlight the effect of adding a concept layer within a neural network. (2) A CNN-LSTM with a Neural Arithmetic Logic Unit (NALU), and (3) a CNN-LSTM with a Neural Accumulator (NAC). These baselines are chosen given they are specifically tailored to arithmetic tasks. For the Hitting Set tasks, we instead use a CNN-LSTM alongside the two CBM variants, since the CNN-LSTM can accept sequence input. The CNN-LSTM in the Hitting Set tasks uses the same CNN encoder as NSIL, applied to each image in the input sequence, followed by an LSTM layer of tunable size, a linear layer of size 1, and a sigmoid layer. The CBM and CBM-S baselines have similar architectures, consisting of the same CNN as NSIL, applied to each image separately. In both variants, the CNN is followed by 3 linear layers where the size of the first two layers are tunable and the last layer is of size 19 in the Arithmetic tasks, and 1 in the Hitting Set tasks. In the Addition task, ReLU activation is used after the first two linear layers for the CBM and CBM-S baselines, as this resulted in improved performance. In the other tasks, no non-linearity was used. The CNN-LSTM-NALU and CNN-LSTM-NAC baselines contain the same CNN as NSIL and the other baselines, followed by an LSTM layer of tunable size, and then either an NALU or NAC layer. Finally, we use the stochastic gradient descent optimiser for all neural networks (Ruder, 2016).

NSIL Configuration. In the Cumulative Arithmetic tasks, we use the same setup as (Dai and Muggleton, 2021), although use ILASP as our symbolic learner. In all other tasks, the neural network in NSIL is the MNIST CNN from (Manhaeve et al., 2018), which consists of an encoder with 2 convolutional layers with kernel size 5, and output sizes 6 and 16 respectively. Each convolutional layer is followed by a max pooling layer of size 2, stride 2. The encoder is followed by 3 linear layers of size 120, 84, and 10 (in the hitting set tasks, the last layer is of size 5), and all layers are followed by a ReLU activation function. Finally, a softmax layer returns the output probability distribution. We use the ILASP symbolic learner, as ILASP enables the search space to be expressed using meta-rules. This ensures a fair comparison to *MetaAbd*, which requires a meta-rule specification to define the inductive search space. For the Hitting Set tasks, we also use ILASP, since ILASP can learn choice rules which are required to solve these tasks. For the Two-Digit Arithmetic tasks, we use a different learning system for ASP, called FastLAS. This demonstrates NSIL is flexible and can support different symbolic learners. FastLAS has increased efficiency in larger search spaces for this specific class of problems. To evaluate NSIL, we compare the final prediction to the ground truth label. Each experiment is repeated 20 times

using 20 randomly generated seeds, and we measure the mean classification accuracy per training epoch, as well as the standard error over the 20 repeats, where 1 epoch = 1 NSIL iteration for comparison to the neural baselines. Each experimental repeat is allowed to run for a maximum of 24 hours. All experiments were run on a shared cluster with the following specification: RHEL 8.5 x86 64 with Intel Xeon E5-2667 CPUs (10 cores total), and 1 NVIDIA A100 GPU, 150GB RAM. The ASP encodings of the domain knowledge are presented in Appendix A.

Hyper-parameters. We perform hyper-parameter tuning for all methods using the Ax library² which adopts a Bayesian Optimisation strategy. All parameters and settings to Ax are left as default. We return the hyper-parameters which yield the highest task accuracy on a held-out validation set after 2 iterations for NSIL, and 5 epochs for the baselines, since these neural models are faster than NSIL. For all models, we tune the learning rate and momentum of the optimiser between the bounds 0.0001 and 0.1, and 0 and 1 respectively. For both CBM models, we tune the number of neurons in the MLP between the bounds 32 and 128 for the first layer, and 64 and 256 for the second layer. For the CNN-LSTM-NAC and CNN-LSTM-NALU in the Two-Digit Arithmetic task, we tune the hidden dimension of the NAC or NALU between the bounds 2 and 6, and the LSTM hidden dimension between the bounds 32 and 256. In the Cumulative Arithmetic task, we use the published hyper-parameters for *MetaAbd* (Dai and Muggleton, 2021). In the hitting sets tasks, we tune the number of neurons in the LSTM hidden layer in the CNN-LSTM between the bounds 32 and 256. We present the hyper-parameters used for all of our models in the following sub-sections, when evaluating each task.

Ablations and Analysis. We also perform two key ablation studies: (1) Where we assume the symbolic knowledge is given, and (2) assume the neural network is pre-trained with concept labels. NSIL has equivalent performance in these cases, whilst *learning* symbolic knowledge, and, training *without* concept labels. Finally, we demonstrate the effectiveness of the novel corrective examples introduced in this chapter, by presenting NSIL results *without* corrective examples. Let us now present our experimental results.

4.3.2 Cumulative Arithmetic

We first consider the Cumulative Addition and Product tasks from (Dai and Muggleton, 2021). The dataset $\mathcal{D}_{\text{task}}$ contains $\langle X, M, y \rangle$ data points where X is a sequence of MNIST digit images, M is empty, and y is the cumulative addition or product of the digits in X . There are $k = 10$ possible values for each digit image. The goal is to train the neural network to classify the digit images in X , and learn a recursive hypothesis that defines the cumulative arithmetic operation. The training set contains 3,000 data points of sequences of length 2–5, and three test sets with sequences of lengths 5, 10, and 100 in the Addition task, and 5, 10, and 15 in the Product task, to verify the trained network and learned hypothesis can extrapolate to longer inputs. Each test set contains 10,000 data points. In terms of the hyper-parameters used, Table 4.1 presents the best performing values obtained.

The background knowledge for ILASP contains comparable relations to that of *MetaAbd*. First, we encode an ASP representation of a list, to enable ILASP to learn recursive programs. Second, we encode the knowledge of the same arithmetic functions, namely, addition, multiplication, and equality. These are recursive encodings to enable the functions to operate over lists. We also include a constraint to ensure there is only one possible output of the program, given a list of input digits. Note this is implicitly available in the dataset, since each data point contains a single label. As for the hypothesis search space, we replicate the same meta-rules as

²<https://ax.dev/>

Task	Learning Rate	Momentum
Addition	0.00103	0.02784
Product	0.00010	0.79772

Table 4.1: NSIL hyper-parameters used in the Cumulative Arithmetic tasks.

used in (Dai and Muggleton, 2021) although express them in an ASP form. The full ASP listing of the background knowledge and search space is presented in Appendix A. To enable NSIL to learn an initial candidate hypothesis, we give examples to ILASP with a full choice over the digit values, for all possible labels with lists of length 2–4. For example, given a data point with the label $y = 10$, and list length 3, we create the following WCDPI:

```

 $e_{\text{pen}} = \infty$ 
 $e_{\text{inc}} = \{ \text{result}(10) \}$ 
 $e_{\text{exc}} = \emptyset$ 
 $e_{\text{ctx}} = \{$ 
  1 {digit(1,0) ; ... ; digit(1,9)} 1.
  1 {digit(2,0) ; ... ; digit(2,9)} 1.
  1 {digit(3,0) ; ... ; digit(3,9)} 1.
  start_list((A,(B,(C,empty)))) :- digit(1,A),digit(2,B),digit(3,C).
 $\}$ 

```

The infinite weight penalty ensures ILASP must cover this example. The label $y = 10$ is encoded in the inclusion set. The exclusion set remains empty, since, the background knowledge contains a constraint to ensure only one possible label is covered per example. The context contains three choice rules, one for each digit in the list, and this denotes the choice over the possible values for each digit (0–9). Finally, the context also contains a `start_list` rule which maps the digit choices into a list format. The special fact `empty` is defined in the background knowledge, and is used to denote the base case of a single element list. The corrective examples are structured similarly to the initial examples, except they do not contain choice rules. Let us assume the Addition sub-task, $X = \langle \text{1, 3, 4} \rangle$, $M = \langle \rangle$, $y = 8$, and $Z = \langle 1, 3, 4 \rangle$. A corrective example pair for this data point is shown in Figures 4.2a and 4.2b:

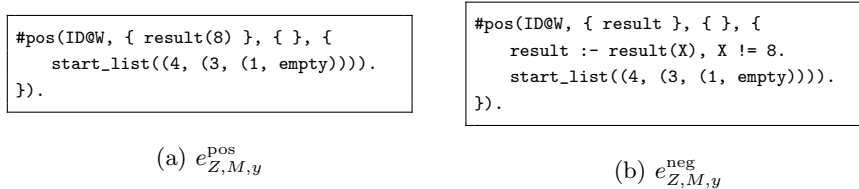


Figure 4.2: Cumulative Addition corrective examples

where ID, and W refer to the example identifier (e_{id}), and the example weight (e_{pen}) respectively. In Figure 4.2a, the digit values are encoded directly into the list representation within the example context (e_{ctx}), and the label $y = 8$ is specified in the inclusion set (e_{inc}). In Figure 4.2b, the same list encoding is used for the digits, but the inclusion set contains the `result` fact, which is true if a different result than 8 is obtained for these digits.

As in (Dai and Muggleton, 2021), we evaluate the neural network accuracy (**Acc.**) on

Seq. Len.	Addition				Product			
	Acc.	MAE			Acc.	log MAE		
	1	5	10	100	1	5	10	15
<i>Meta_Abd</i>	0.953	0.510	1.299	6.587	0.977	0.334	0.495	2.374
NSIL	0.984	0.208	0.550	3.960	0.983	0.291	0.469	2.098

Table 4.2: Cumulative Arithmetic task results.

Addition
 $f(A,B) :- eq(A,B).$
 $f(A,B) :- add(A,C), f(C,B).$

Product
 $f(A,B) :- eq(A,B).$
 $f(A,B) :- mult(A,C), f(C,B).$

Figure 4.3: Cumulative Arithmetic learned hypotheses

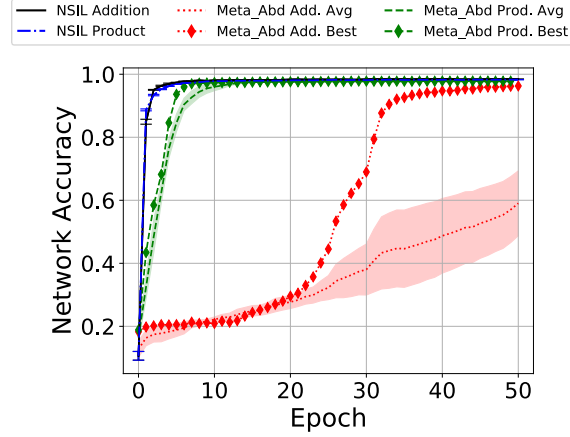


Figure 4.4: Cumulative Arithmetic network accuracy

predicting single digits, and the mean average error (**MAE**), and log mean average error (**log MAE**) when extrapolating to longer inputs in the addition and product tasks respectively. For this experiment only, we report results over 5 repeats to match (Dai and Muggleton, 2021). The results are presented in Table 4.2, and Figures 4.3 and 4.4.

In both tasks, NSIL learns the correct recursive knowledge (Figure 4.3), and outperforms *Meta_Abd* (Table 4.2). Also, NSIL has a higher neural network accuracy (Figure 4.4), closer to the state-of-the-art accuracy of 0.9987 when a CNN is trained in a fully supervised fashion (Byerly et al., 2021), despite NSIL being trained without concept labels. NSIL trains the neural network to its maximal accuracy with significantly less training epochs than *Meta_Abd*. *Meta_Abd* relies upon computing the exact pseudo-labels for each image in order to train the network accurately, whereas in NSIL, NeurASP can train the network with multiple digit choices for each image using a semantic loss function (Xu et al., 2018). This helps the network achieve a higher accuracy during the earlier stages of training. In Figure 4.4, the error bars and shaded regions indicate standard error over the 5 repeats. We plot the *Meta_Abd* Addition results from their GitHub repository, and re-run their Product task with 5 randomly generated seeds as these results are not available. We also plot the best *Meta_Abd* result compared to the average, to demonstrate that *Meta_Abd* can be highly sensitive to the random seed used to initialise the network in tasks with dense abductive spaces. Nonetheless, the average NSIL performance outperforms *Meta_Abd* in their best cases, on both tasks.

4.3.3 Two-Digit Arithmetic

We now consider the MNIST Addition task for two single digit numbers presented in (Manhaeve et al., 2018), and a new variation called MNIST Even9Plus (E9P), that requires learning negation as failure, which is not supported by *Meta_Abd*. In this task, the same pair of images are used,

but the label is equal to the digit value of the second image, if the digit value of the first image is even, or 9 plus the digit value in the second image otherwise. The purpose of the E9P task is to demonstrate NSIL can be used to solve multiple tasks within the same domain, i.e., the search space does not have to be engineered *exactly* to solve a particular task. In this case, both sub-tasks are defined over the same set of inputs and outputs, which is more challenging than the Cumulative Arithmetic domain, where the output space is significantly different between the Addition and Product sub-tasks. We include a comparison to *Meta_Abd* on the MNIST Addition task, but note that a significantly reduced search space is used compared to NSIL, as the relations required for E9P are not included, given that *Meta_Abd* does not support learning negation as failure. Both tasks use training, validation and test datasets of size 24,000, 6,000, and 5,000 data points respectively. The relations **even**, **not even**, **plus_nine**, and '=', as well as the function '+' are given in the background knowledge, and are used to specify the search space for the symbolic learner. The **even** relations define even digits, by checking the digit modulo 2 is equal to 0, and the **plus_nine** relation defines 9+ a given digit. We also include a similar constraint to the Cumulative Arithmetic task, which ensures there can only be one possible output for a given pair of input digits. The full ASP listing of the background knowledge and search space is given in Appendix A. To construct the initial hypothesis learning task for the symbolic learner, we give examples to FastLAS with a full choice over the digit values, for each possible label 0–18. For example, given a data point with the label $y = 10$, we create the following WCDPI:

$$\begin{aligned}
 e_{\text{pen}} &= \infty \\
 e_{\text{inc}} &= \{ \text{result}(10) \} \\
 e_{\text{exc}} &= \emptyset \\
 e_{\text{ctx}} &= \{ \\
 &\quad 1 \{ \text{digit}(1,0) ; \dots ; \text{digit}(1,9) \} 1. \\
 &\quad 1 \{ \text{digit}(2,0) ; \dots ; \text{digit}(2,9) \} 1. \\
 &\quad \}
 \end{aligned}$$

The infinite weight penalty ensures FastLAS must cover this example. The exclusion set remains empty, since, the background knowledge contains a constraint to ensure only one possible label is covered per example. The context contains two choice rules, one for each digit, and this denotes the choice over the possible values for each digit (0–9). Note the index of each digit is implicitly available given the image sequence X . For the corrective examples, let us assume the E9P sub-task, $X = \langle \text{2}, \text{7} \rangle$, $M = \langle \rangle$, $y = 1$, and $Z = [2, 1]$. A corrective example pair for this data point is shown in Figures 4.5a and 4.5b.

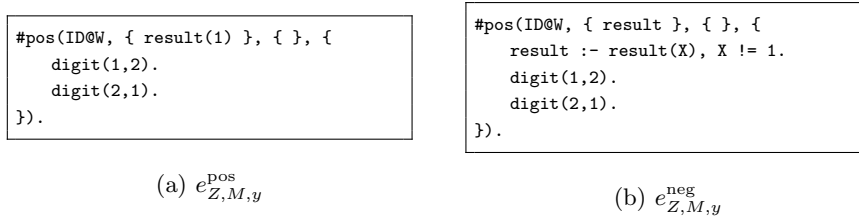


Figure 4.5: E9P corrective examples

ID, and W refer to the example identifier (e_{id}), and the example weight (e_{pen}) respectively. For

Method	Addition		E9P	
	Learning Rate	Momentum	Learning Rate	Momentum
NSIL	0.00054	0.44685	0.00219	0
CNN	0.00123	0.14936	0.00066	0.41157
CBM	0.00229	0	0.00161	0
CBM-S	0.00103	0.35808	0.00062	0.83642
CNN-LSTM-NAC	0.00161	0	0.00161	0
CNN-LSTM-NALU	0.00161	0	0.00161	0

Table 4.3: Learning rate and momentum hyper-parameters used in the Two-Digit Arithmetic tasks.

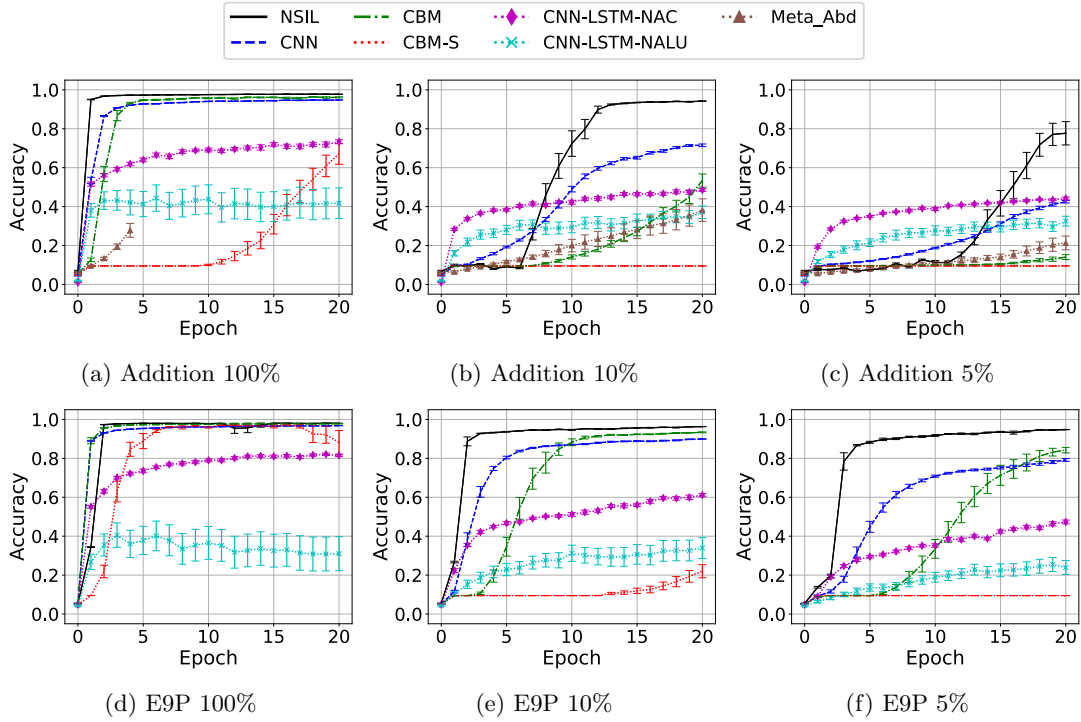


Figure 4.6: Two-Digit Arithmetic results with reduced training sets.

the hyper-parameters, Table 4.3 shows the learning rate and momentum values obtained for all methods, to 5 decimal places. For the CBM and CBM-S models, the number of neurons in layers 1 and 2 of the MLP are 39 and 120, and 110 and 71 respectively. For the CNN-LSTM-NAC and the CNN-LSTM-NALU, the number of neurons in the arithmetic hidden layers is 2, and the number of neurons in the LSTM hidden layers is 128, for both models.

The overall accuracy results are shown in Figure 4.6 for reducing percentages of training data (100%–5%), and the hypotheses learned by NSIL are shown in Figure 4.7. NSIL outperforms the neural baselines on both tasks for all dataset percentages, and learns the correct hypotheses. The `result` predicate maps input digits `V0` and `V1` to the result `V2`. You can see the correct addition rule has been learned, and in the E9P task, the result is equal to the second digit (`V1`), if the first digit (`V0`) is even. If `V0` is not even, the result is equal to 9 plus `V1`. In terms of the end-to-end task accuracy, on the Addition task, NSIL achieves superior performance in a much larger search space compared to *Meta_Abd*, which also times out after iteration 4 with 100% of the data. NSIL trains the neural network to classify MNIST images with a mean accuracy of 0.989, 0.970, and 0.891, for each dataset percentage respectively, which in the 100% case is within 1%

```

Addition
result(V0,V1,V2) :- V2 = V0 + V1.

E9P
result(V0,V1,V2) :- even(V0), V2 = V1.
result(V0,V1,V2) :- not even(V0), plus_nine(V1,V2).

```

Figure 4.7: Two-Digit Arithmetic learned hypotheses

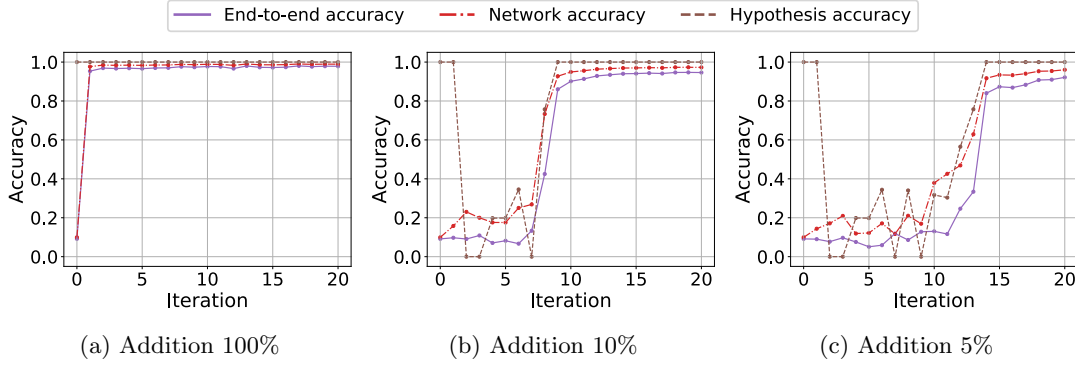


Figure 4.8: Two-Digit Addition accuracy breakdown.

of the performance of a fully supervised model (0.9987) (Byerly et al., 2021). For the E9P task, the mean network accuracy is 0.986, 0.974, and 0.962, for each dataset percentage respectively. NSIL has greater performance in the E9P task than in the Addition task, because the label is more informative, despite the same domain knowledge and search space. There is a reduced number of possible concept values for certain labels, which gives a stronger back-propagation signal for training the network. For example, for all data points with label $y = 10$, the first digit must be odd and the second digit must be 1, as it is only possible to obtain 10 if the first digit is odd, and $9 +$ the second digit sums to 10. In the addition task, it is possible to obtain 10 with multiple combinations of the first and second digits.

It is interesting to explore the effect of the corrective examples on NSIL’s performance, especially in reduced percentages of training data. Figures 4.8 and 4.9 present a breakdown of NSIL’s performance in terms of the accuracy of the neural network, the learned hypothesis, and the end-to-end accuracy at each iteration. For this analysis, we consider the first experimental repeat only, given NSIL has very low standard error across the 20 repeats. In Figure 4.8, the initial learning task is able to learn the correct addition hypothesis, as it is the shortest hypothesis in the search space that covers the initial examples (with a choice of digits). This is shown by the perfect accuracy of the learned hypothesis at the first iteration. In the case of 100% training data (Figure 4.8a), the network is trained to a high accuracy, as the hypothesis is correct. This enables NSIL to retain the hypothesis on the next iteration, as positive corrective examples for correct $\langle Z, M, y \rangle$ tuples have high weight penalty. For example, inspecting the learning task for the symbolic learner at iteration 2, the corrective example pair for $Z = \langle 3, 5 \rangle$, $M = \langle \rangle$, and $y = 8$ is as follows:

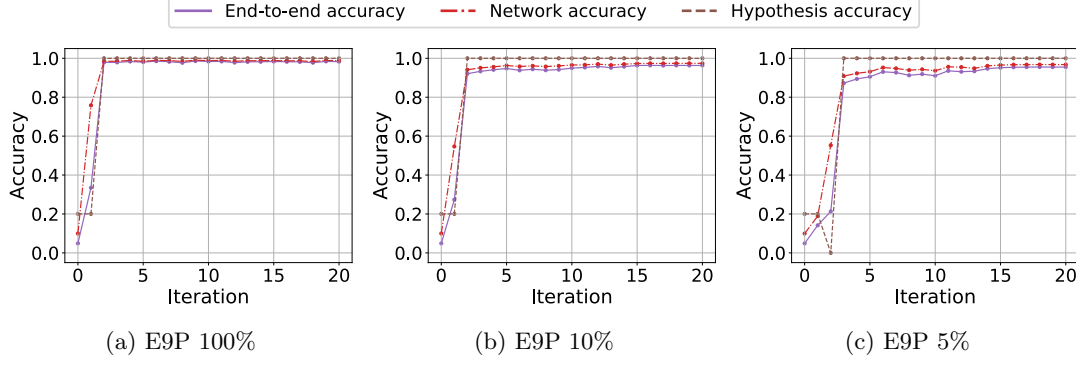
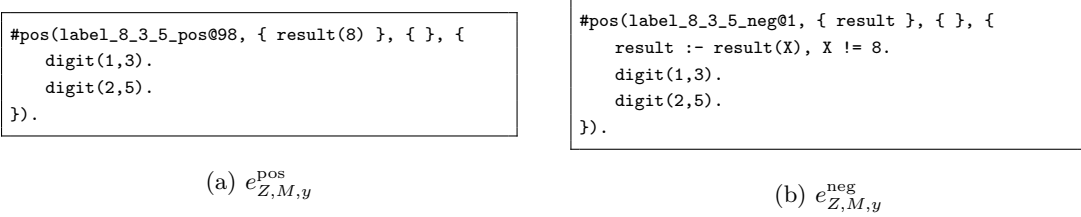


Figure 4.9: Two-Digit E9P accuracy breakdown.

Figure 4.10: Corrective examples for $Z = \langle 3, 5 \rangle$, $M = \langle \rangle$, and $y = 8$, Addition 100% task, iteration 2.

You can see the positive corrective example has weight 98, and the negative corrective example has weight 1. This provides a strong bias to the symbolic learner to learn a hypothesis to cover the positive case on the next iteration, for this particular corrective example. This can be viewed as exploitation of the current hypothesis. These weights are obtained, because the FNR for label 8 is 4.4077 after the first iteration, indicating $\sim 96\%$ of data points with label 8 are predicted correctly. When calculating the corrective example weights based only on the FNR, $W(e_{\langle 3,5 \rangle, 8}^{\text{pos}}) = 0$, $W(e_{\langle 3,5 \rangle, 8}^{\text{neg}}) = \frac{4.4077}{9} = 0.490$ which is rounded to 0 as the nearest integer. Note that the FNR is divided by 9 because there are 9 possible answer sets of $\Pi_{\langle M, y \rangle}$, i.e., 9 possible ways of obtaining $y = 8$ given the correct hypothesis. At this stage, there is no bias given to the symbolic learner, since both weights are equal. Now, when the neural network predicts $Z = \langle 3, 5 \rangle$ for examples with $M = \langle \rangle$, and $y = 8$, it predicts on average with $\sim 97\%$ confidence. Therefore, $W(e_{\langle 3,5 \rangle, 8}^{\text{pos}}) = 0 + 97 = 97$, and $W(e_{\langle 3,5 \rangle, 8}^{\text{neg}}) = 0 - 97 = -97$ which is clipped to 0. Since we shift the weights to the range $\{1..101\}$ for the LAS symbolic learner, this gives us the weights $W(e_{\langle 3,5 \rangle, 8}^{\text{pos}}) = 98$, and $W(e_{\langle 3,5 \rangle, 8}^{\text{neg}}) = 1$. Since the correct hypothesis is retained at the next iteration, the neural network is able to be trained to a high accuracy, and the overall accuracy remains high throughout training.

It is also interesting to analyse the 10% case for Addition (Figure 4.8b), since NSIL requires more iterations to converge to a higher end-to-end accuracy. This is also the case when 5% of the training data is used (Figure 4.8c). As in the 100% case, NSIL learns the correct hypothesis on the first iteration, because it is the shortest hypothesis in the search space that covers the initial examples. However, due to the limited training data, and because we train the neural network for only 1 epoch on each NSIL iteration, the neural network confidence is not high enough to provide a positive signal, nor is it necessarily allocated to the correct predictions. Also, due to the neural network's under performance, the FNR per label is high, and therefore there is a higher negative signal for the corrective examples. This encourages exploration of the hypothesis

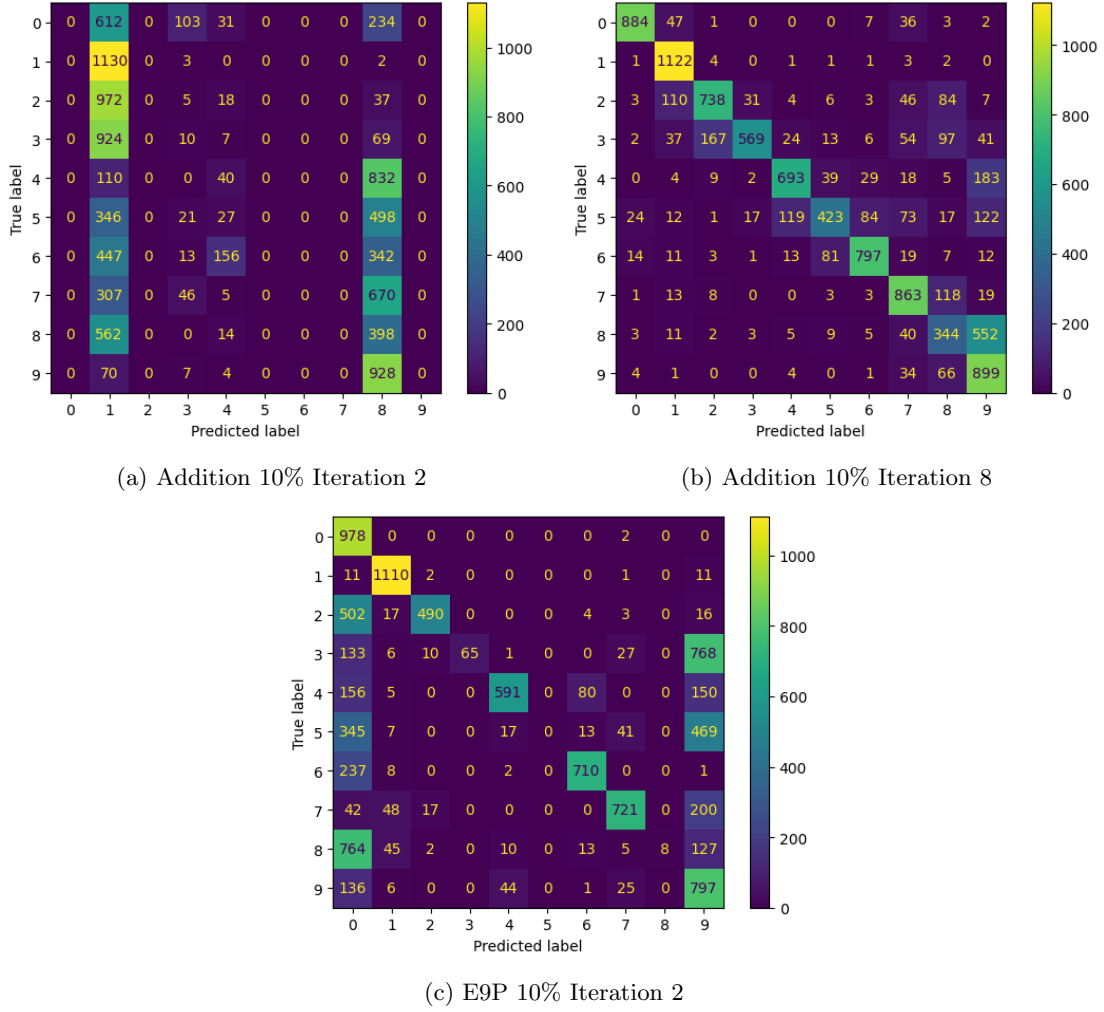
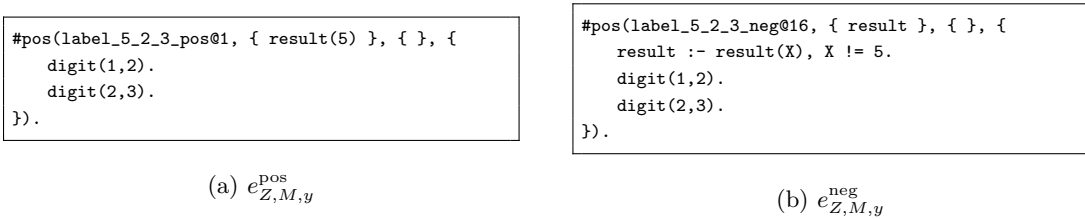


Figure 4.12: Neural Network confusion matrices

search space. Consider the corrective example pair associated with $Z = \langle 2, 3 \rangle$, $M = \langle \rangle$, and $y = 5$ at iteration 2, when 10% of the training is used:

Figure 4.11: Corrective examples for $Z = \langle 2, 3 \rangle$, $M = \langle \rangle$, and $y = 5$, Addition 10%, iteration 2.

The FNR for label 5 is 91.27, so the weights are initially set to $W(e_{\langle 2,3 \rangle, 5}^{\text{pos}}) = 0$, and $W(e_{\langle 2,3 \rangle, 5}^{\text{neg}}) = \frac{91.27}{6} \approx 15.21$, which is rounded to 15. Now, looking at the confusion matrix for the neural network predictions in Figure 4.12a, it is clear the neural network never predicts the digit 2. Therefore, there is no positive weight update based upon the neural network confidence score, since $D_{\langle Z, M, y \rangle}$ is empty. This explains why the negative weight in Figure 4.11b is 16 (since we add 1 to the weights for the LAS symbolic learner), and the positive weight is 1. This encourages the symbolic learner to learn a different hypothesis on the next iteration and explore the hypothesis search space. On subsequent iterations this continues, as the symbolic

	Domain knowledge	Convergence iteration	Convergence time (s)
*	+, =, even, not even, plus_nine	2	475.44
	+, -, ×, ÷, =, even, not even, plus_nine	2	547.81
	=, even, not even, plus_nine, ..., plus_one	1	<i>402.03</i>
	+, =, even, not even, plus_nine, ..., plus_one	2	<i>881.35</i>
	+, -, ×, ÷, =, even, not even, plus_nine, ..., plus_one	2	<i>57869.12</i>

Table 4.4: Increasing the search space for the symbolic learner in the E9P task. * denotes the configuration used in the main results. Run-times in italics indicate that weight pruning is used to remove corrective examples with weight < 5 , to obtain results efficiently.

learner explores the search space. On iteration 9, the correct hypothesis is learned. This is because training the network with sub-optimal hypotheses in earlier iterations is sufficient to predict some digits correctly. For example, Figure 4.12b presents the confusion matrix after neural network training at iteration 8. Whilst there is not a perfect diagonal, you can clearly see some digits are predicted correctly. This is sufficient to generate corrective example pairs with a high positive signal for the correct $\langle Z, M, y \rangle$ combinations, and thus provide a stronger bias to the symbolic learner to enable the correct hypothesis to be learned. This hypothesis is then retained for the remainder of training, since the neural network is more accurate.

For the E9P task, the correct hypothesis is not learned on the first iteration, because the addition hypothesis is shorter, and covers the initial examples with a valid choice of digits. However, the corrective examples enable NSIL to converge to the correct hypothesis in later iterations, and in the case of 10% and 5% training data, sooner than in the Addition task (see Figures 4.9b and 4.9c). This is because the E9P task results in fewer answer sets of the $\Pi_{\langle M, y \rangle}$ program for certain examples, which gives a stronger signal for training the neural network. Figure 4.12c presents the neural network confusion matrix after the 2nd NSIL iteration with 10% of the training data. Although the neural network often predicts digits 0 and 9, there are still enough digits predicted correctly to generate an appropriate bias for the symbolic learner through the corrective example weights. To summarise, our novel contribution of biasing the symbolic learner with corrective examples, enables NSIL to balance exploration vs exploitation of the hypothesis search space. We have demonstrated convergence to the correct hypotheses in challenging situations where the neural network is under-trained due to limited training data.

Now, we investigate whether NSIL can converge to the correct hypothesis given an increased search space with different arithmetic operators. Considering the E9P task, which is harder because the correct knowledge is not learned on the first iteration, we increase the search space with the additional functions; '-', '×', '÷', as well as `plus_eight, ..., plus_one` relations. As shown in Table 4.4, NSIL converges to the correct hypothesis in all of these cases, but requires more time to do so, as the size of the search space increases.

To highlight the effectiveness of each component, we now present three ablations of the NSIL framework: (1) We remove the requirement to learn the symbolic knowledge and assume the knowledge is given. (2) We pre-train the neural network with manually annotated concept labels, and (3) we evaluate NSIL without corrective examples, in order to demonstrate the impact of

Dataset %	Addition			E9P		
	100	10	5	100	10	5
FF-NSL	0.9753 (0.0021)	0.9362 (0.0029)	0.9151 (0.0058)	0.9809 (0.0016)	0.9513 (0.0030)	0.9346 (0.0051)
NeurASP	0.9762 (0.0013)	0.9492 (0.0016)	0.9149 (0.0051)	0.9797 (0.0015)	0.9642 (0.0009)	0.9500 (0.0014)
NSIL	0.9762 (0.0013)	0.9449 (0.0025)	0.8782 (0.0134)	0.9816 (0.0009)	0.9634 (0.0007)	0.9510 (0.0016)

Table 4.5: Two-Digit Arithmetic FF-NSL and NeurASP comparison. Best results highlighted in bold (we declare a tie in the case of overlapping error bars).

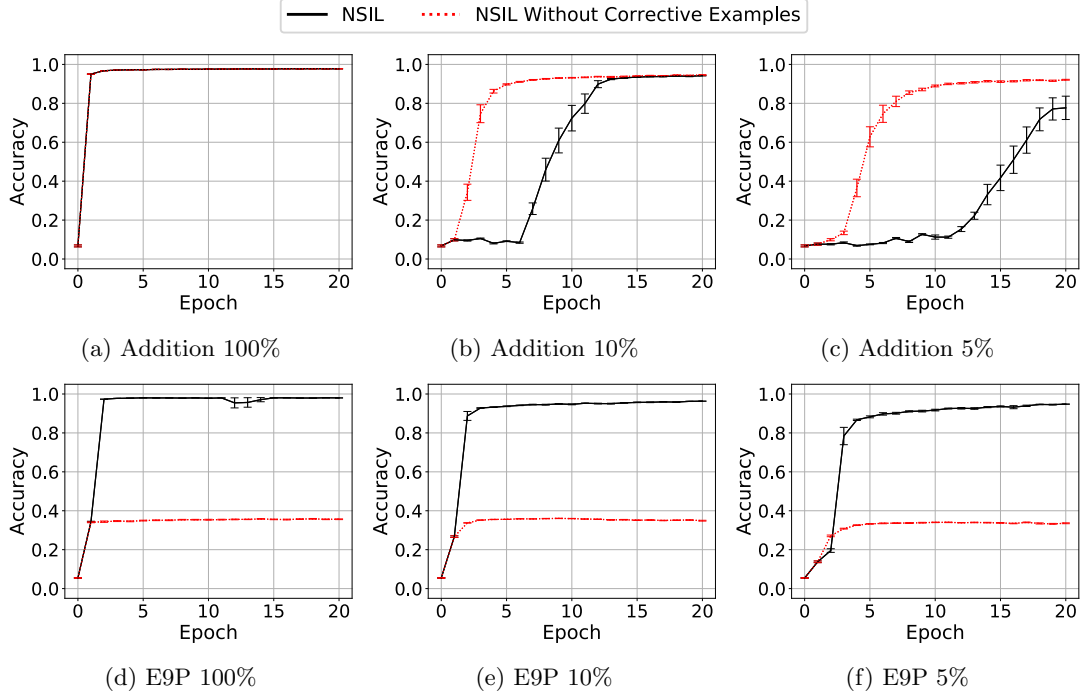


Figure 4.13: Two-Digit Arithmetic accuracy comparison with no corrective examples. Average over 20 repeats.

the novel contribution of this chapter. To implement the first two ablations, we respectively evaluate NeurASP (Yang et al., 2020), which assumes the symbolic knowledge is given, and FF-NSL (Cunnington et al., 2023a), which pre-trains the neural network on a separate dataset with concept label information. FF-NSL then uses the neural network predictions to train the symbolic learner. We provide a comparison to NSIL, and train each method for 20 epochs (where 1 epoch = 1 NSIL iteration). For the first two ablations, we evaluate the mean performance and standard error over 5 repeats. The results are presented in Table 4.5 for reducing percentages of training data (100% — 5%). The results indicate the final task accuracy after performing a forward pass on the neural and symbolic components. In both the Addition and E9P tasks, NSIL achieves comparable accuracy to the fully supervised approach of FF-NSL, even when *no concept labels* are observed during training. Also, NSIL achieves comparable accuracy to NeurASP, whilst also *learning* the symbolic knowledge. For the final ablation study, we compare the main NSIL results from Figures 4.6 to NSIL without corrective examples. The results are averaged over 20 repeats, and are presented in Figure 4.13.

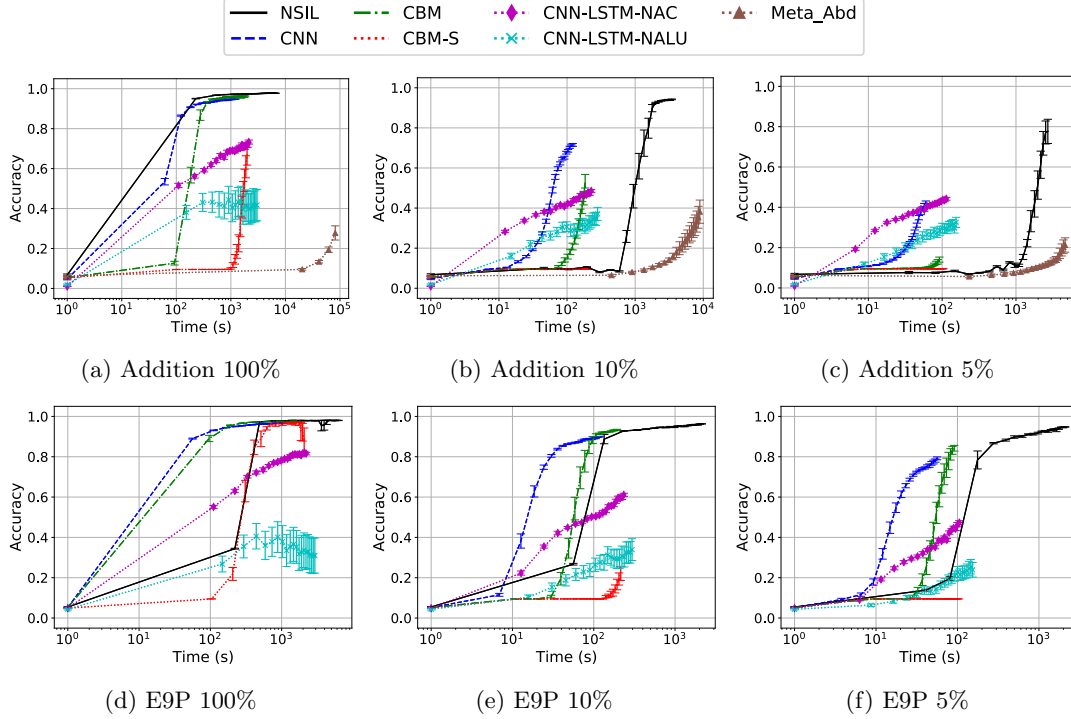


Figure 4.14: Two-Digit Arithmetic learning time vs. accuracy with reducing training set sizes.

The corrective examples enable NSIL to converge to a more accurate solution in most cases, by biasing the symbolic learner to explore the symbolic search space. This is evidenced in the E9P task shown in Figures 4.13d-4.13f, where the correct knowledge is *not* learned on the first iteration. Without corrective examples, NSIL retains the knowledge learned on the first iteration throughout training. In cases where the correct knowledge *is* learned on the first iteration during the initial bootstrap phase, the effect of the corrective examples depends on the size of the dataset. When there are sufficient data points to train the neural network accurately, the corrective examples do not offer any improvement, because the network can be trained given the correct knowledge (see Figure 4.13a). For small datasets, the corrective examples together with a sub-optimal network may lead to a delay in convergence compared to the case where there are no corrective examples (see Figures 4.13b and 4.13c). However, in real-world applications, we consider these to be very special cases, as it is unlikely the correct knowledge will be learned on the first iteration. Therefore, corrective examples are important to NSIL’s performance.

Finally, Figure 4.14 show the learning time vs. accuracy comparison between NSIL and the baseline methods. For each method, we plot the time taken to complete 20 epochs, and each point shows the accuracy after an epoch of training (1 epoch = 1 NSIL iteration). Error bars indicate standard error. NSIL requires more time to complete 20 epochs, but achieves a greater accuracy than the neural methods. Also, NSIL has comparable learning time to $Meta_{Abd}$ on the addition tasks.

4.3.4 The Hitting Set Problem

To verify NSIL can converge to the correct hypothesis, and train the neural network accurately in other domains, we consider two variations of the Hitting Set problem (Karp, 1972). The first is the standard formulation: Consider a universe of elements U , and a collection of sets S , which contains subsets of U . A hitting set is a special set $\Phi \subseteq U$ that intersects with every subset in

S . Given S and an integer κ , the task is to decide if there is a hitting set of S with size $\leq \kappa$.

Example 4.3.1. Let $U = \{1 \dots 5\}$, $S_1 = \{\{1\}, \{1, 3\}, \{4\}\}$, and $\kappa = 2$. S_1 has a hitting set of size ≤ 2 : $\Phi = \{1, 4\}$. We say that each subset in S is “hit” if it intersects with the hitting set. Now consider $S_2 = \{\{1\}, \{2, 3\}, \{4\}\}$. S_2 has no hitting set of size ≤ 2 .

In our experiments, we replace elements $\{1 \dots 5\}$ in U with corresponding MNIST images from classes 1-5 (i.e., $\mathcal{Z} = \{1 \dots 5\}$). We assume $\kappa = 2$, and we construct $\langle X, M, y \rangle$ data points where X represents a sequence of elements based upon a collection S , M denotes the subset identifier of each image in S , and y is a binary label indicating whether S has a hitting set of size $\leq \kappa$. We also assume a maximum of 4 elements, S is well-formed, with no duplicate elements in a set, and elements within the same set are arranged in ascending order. For example, if $S = \{\{\text{7}\}, \{\text{1}, \text{3}\}, \{\text{4}\}\}$, $X = \langle \text{7}, \text{1}, \text{3}, \text{4} \rangle$, $M = \langle 1, 2, 2, 3 \rangle$, and $y = 1$. We denote this standard hitting set variant as HS. To demonstrate NSIL can solve multiple tasks within the same domain, we introduce a second variant, denoted CHS, which adds the constraint that no hitting sets occur if $|S| \geq 3$, and element 1 (in image form) exists in any subset in S . We also present results for both variants with a more difficult perception task using FashionMNIST images instead of MNIST. The goal is to train the neural network to classify images, whilst learning the HS or CHS rules to decide the existence of a hitting set. Training, validation and test datasets contain 1502, 376, and 316 examples respectively. Each concept value is represented in ASP as `ss_element(s, z)`, where $s \in \{1 \dots 4\}$ is a subset identifier, and $z \in \{1 \dots 5\}$ is the associated element. The domain knowledge B contains $\kappa = 2$, the relation ‘!=’ and the relation `hs`, which defines an element of the hitting set. B also contains the relations for the CHS task: `ss_element(3, V1)` and `ss_element(V2, 1)` that define respectively subset 3 (and any element V1), and element 1 in any subset V2. The full listing of the background knowledge is presented in Appendix A.

In these tasks, we use a CNN-LSTM baseline in addition to the CBM variants, where a CNN first encodes a feature vector for each image in a sequence X , and an LSTM returns a binary classification as to the existence of a hitting set. To encode the subset structure of S for the CNN-LSTM, we create an image of ‘{’ and ‘}’ characters and pass in the entire sequence as input. Note that in NSIL, the structure is passed directly into the symbolic learner using the `ss_element` relations.

To construct the initial learning task for ILASP, we create a set of examples that enable ILASP to cover each possible combination of meta-data and target label. In the Hitting Set domain, the meta-data contains the subset identifiers for each image, which corresponds to the structure of the subsets in S . To achieve the desired behaviour of the initial learning task, we give all possible combinations of corrective examples to ILASP, with equal weights, and we use the meta-level injection feature to force at least one example with a certain meta-data and label to be covered. This guarantees that on the first iteration, ILASP will learn a hypothesis that covers each meta-data and target label combination, with a valid choice of digit values. Recall that we assume the examples are well-formed, where no duplicate elements appear in the same subset, and the digits within the same set are arranged in ascending order. Consider the collection $S = \{\{\text{7}\}, \{\text{1}, \text{3}\}, \{\text{4}\}\}$, and the associated data point $X = \langle \text{7}, \text{1}, \text{3}, \text{4} \rangle$, $M = \langle 1, 2, 2, 3 \rangle$, and $y = 1$. The initial examples for this combination are:

Method	MNIST Images		FashionMNIST Images	
	Learning Rate	Momentum	Learning Rate	Momentum
NSIL	0.00082	0.76434	0.00219	0
CNN-LSTM	0.00245	0.87474	0.00633	0.69360
CBM	0.00237	0.85093	0.00372	0.78103
CBM-S	0.00489	0.71138	0.01476	0.57376

Table 4.6: Learning rate and momentum hyper-parameters used in the Hitting Set tasks.

<pre>#inject("group(group_4,template_2_1_1_3_4_pos)."). #inject("weight(template_2_1_1_3_4_pos,1)."). #pos(template_2_1_1_3_4_pos, { }, { }, { ss_element(1,1). ss_element(2,1). ss_element(2,3). ss_element(3,4). }).</pre>	<pre>#inject("group(group_5,template_2_1_1_3_4_neg)."). #inject("weight(template_2_1_1_3_4_neg,1)."). #neg(template_2_1_1_3_4_neg, { }, { }, { ss_element(1,1). ss_element(2,1). ss_element(2,3). ss_element(3,4). }).</pre>
--	--

(a) $e_{Z,M,y}^{\text{pos}}$ (b) $e_{Z,M,y}^{\text{neg}}$

Figure 4.15: Hitting Set corrective example combination used during the initial ILASP learning task.

The example contexts define the `ss_element` relations, which encode the subset ID and the digit value. Since the labels in this task represent a binary classification as to whether the given collection contains a hitting set or not, we represent the positive and negative corrective examples with positive and negative WCDPIs, respectively. The two `#inject` statements define the example's group, and the weight. In meta-level injection, at least one example from each group has to be covered by the learned hypothesis. In the initial learning task, each group contains examples for all valid sequences of digit values of a given subset structure, and there exists a positive group and a negative group for each structure. The example weights are set to 1, to provide no bias to the symbolic learner as to which example for a given sequence of digits should be covered. This yields the desired behaviour for the initial task. On subsequent iterations of NSIL, no new corrective examples need to be added (since they all exist in the initial task), and we simply adjust the weights as usual, based upon the FNR and the neural network confidence.

For the hyper-parameters, Table 4.6 shows the learning rate and momentum values obtained for all methods, to 5 decimal places. For the CNN-LSTM, the number of neurons in the LSTM hidden layer is 112 for MNIST images, and 138 for FashionMNIST images. For the CBM model, the number of neurons in layers 1 and 2 of the MLP are 34 and 204, and 55 and 20, for MNIST and FashionMNIST images respectively. For CBM-S, these values are 62 and 104, and 32 and 94, respectively.

Figure 4.16 shows the overall accuracy for both Hitting Set tasks using MNIST and FashionMNIST images. The learned hypotheses are shown in Figure 4.17. NSIL outperforms the baselines on both tasks and learns the correct knowledge. The first rule in each hypothesis is a choice rule that generates all possible elements of a hitting set. `V1` is the index in the hitting set, and `V2` is the element. Note for each combination of index and element, the choice rule generates 0 or 1 `hs` facts. The second rule in each hypothesis defines what it means for a subset in the collection to be hit, i.e., the subset `V1` is hit, if there is an element in the subset `V2` that is also in the hitting set, at index `V3`. The third rule is a constraint that ensures there are no hitting sets, if there exists a subset `V1` that is not hit, as defined by the second rule. The fourth rule

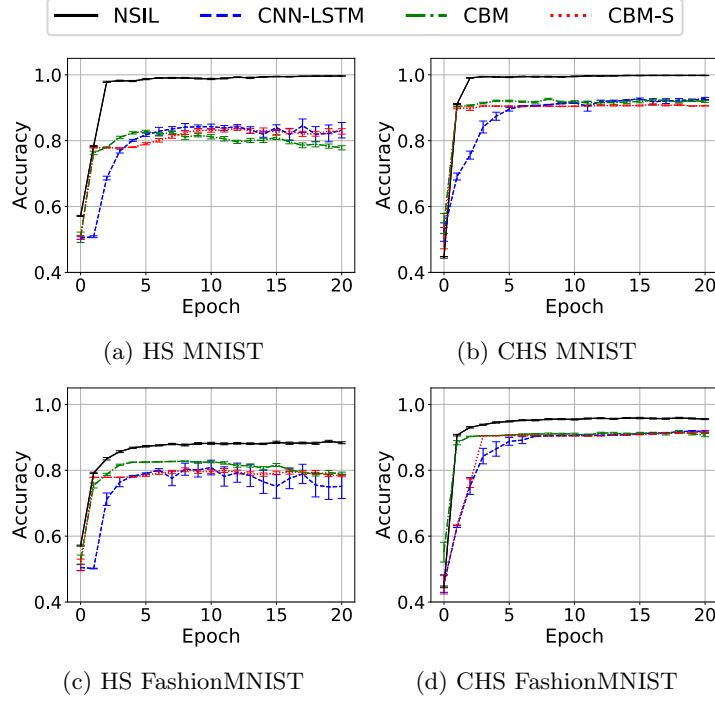


Figure 4.16: Hitting Set accuracy

```

HS
0 {hs(V1,V2) } 1 :- elt(V2), hs_index(V1).
hit(V1) :- hs(V3,V2), ss_element(V1,V2), hs_index(V3), elt(V2), ss(V1).
:- ss_element(V1,V2), not hit(V1), elt(V2), ss(V1).
:- hs(V3,V1), hs(V3,V2), V1 != V2, hs_index(V3), elt(V2), elt(V1).

CHS
0 {hs(V1,V2) } 1 :- elt(V2), hs_index(V1).
hit(V1) :- hs(V3,V2), ss_element(V1,V2), hs_index(V3), elt(V2), ss(V1).
:- ss_element(V1,V2), not hit(V1), elt(V2), ss(V1).
:- hs(V3,V1), hs(V3,V2), V1 != V2, hs_index(V3), elt(V2), elt(V1).
:- ss_element(3,V2), ss_element(V1,1), elt(V2), ss(V1).

```

Figure 4.17: Hitting Set learned hypotheses

is also a constraint that prevents different elements occurring at the same index in the hitting set. Finally, in the CHS task, we learn the extra (desired) constraint that ensures if a collection has 3 subsets, and if the element 1 occurs in any subset in the collection, then there are no hitting sets. In Figure 4.16, all methods perform better on the CHS task than the HS task. This is due to a shortcut; a negative test example with 3 subsets can be correctly predicted if the neural network predicts element 1 for *any* image within the subset, as opposed to having to rely on accurate predictions for more images, when the constraint is not present in the HS task. When classifying images, the neural network is trained to a mean accuracy of 0.993 (MNIST) and 0.897 (FashionMNIST) on the HS task, and 0.993 (MNIST) and 0.894 (FashionMNIST) on the CHS task. Note that the neural network accuracy is for concepts 1-5 only, since our universe of elements $U = \{1 \dots 5\}$.

The knowledge learned by NSIL has multiple answer sets, as there could be multiple hitting sets of a given collection of size $\leq \kappa$ (generated by the choice rule). For example, assuming the correct rules are learned, and $S = \{\{\mathbf{1}, \mathbf{2}\}, \{\mathbf{2}, \mathbf{3}\}\}$, there are four hitting sets of size $\leq \kappa$: $\{2\}$, $\{1, 2\}$, $\{1, 3\}$, and $\{2, 3\}$. Despite training labels indicating the existence of a hitting set, the rules learned by NSIL can generate *all* the hitting sets of a given collection at inference time. We verified this by computing the Hamming loss (Tsoumakas and Katakis, 2007) against the ground

Domain knowledge	Convergence iteration	Convergence time (s)
Standard, ssID 2, ssID 4	1	111.34
Standard, ssID 2, el 2	1	113.21
Standard	1	115.77
Standard, ssID 1, el 4	1	130.71
Standard, ssID 3, el 4	2	734.70
* Standard, ssID 3, el 1	2	852.84
Standard, ssID 1, ssID 3	2	888.54

Table 4.7: Increasing the search space for the symbolic learner in the Hitting Set FashionMNIST task. * denotes the configuration used in the main results.

	HS		CHS	
	MNIST	FashionMNIST	MNIST	FashionMNIST
FF-NSL	0.9937 (0.0017)	0.8816 (0.0110)	0.9962 (0.0012)	0.9563 (0.0034)
NeurASP	0.9981 (0.0013)	0.8975 (0.0041)	0.9994 (0.0006)	0.9538 (0.0070)
NSIL	0.9962 (0.0012)	0.8747 (0.0053)	0.9981 (0.0013)	0.9544 (0.0021)

Table 4.8: Hitting Set ablation results. Best results highlighted in bold (we declare a tie in the case of overlapping error bars).

truth hitting sets for all test data points. The Hamming loss is a multi-label classification metric, and is calculated using the Hamming distance between the ground-truth labels and the predicted labels. The Hamming distance is equal to the number of positions in the predicted vector that are different from the ground-truth vector. The Hamming loss then normalises the Hamming distance against the length of the vector. Therefore, in our case, a Hamming loss of 0 indicates NSIL predicted all of the possible hitting sets for a given example perfectly. We evaluate the mean Hamming loss over the 20 repeats. The results are: 0.003 and 0.150 on the HS task, and 0.003 and 0.126 on the CHS task, for the MNIST and FashionMNIST images respectively. This indicates almost perfect hitting set generation, with the minor errors due to mis-classifications by the neural network, as opposed to errors in the learned knowledge. Also, it is crucial to note that the learned knowledge does not depend on the number of elements, the value of κ , or the possible concept values, despite these restrictions during training. This demonstrates NSIL is able to *generalise* beyond the training setup as the learned knowledge can be applied to a wide range of downstream tasks. Contrast this to the neural baselines, which can only be used during inference for the task used during training.

To extend the search space for ILASP, and explore whether NSIL can converge to the correct hypotheses in these cases, we sampled 6 sets of additional `ss_element` domain relations in the more challenging HS FashionMNIST task. The results are presented in Table 4.7, and NSIL learns the correct hypothesis in all of these cases, but sometimes requires more iterations and/or time to do so. The *Standard* domain knowledge contains only the predicates required to learn the expected rules of the HS task. This includes $\kappa = 2$, the ‘!=’ relation, and the relation `hs`, which defines an element of the hitting set. The additional `ss_element` relations contain a combination of elements {1 .. 5} (denoted ‘el’) and subset identifiers {1 .. 4} (denoted ‘ssID’).

We now present results for the three ablations studied in the Two-Digit Arithmetic Tasks: (1) We remove the requirement to learn the symbolic knowledge, and assume the knowledge is given. (2) We pre-train the neural network with manually annotated concept labels, and (3) we evaluate

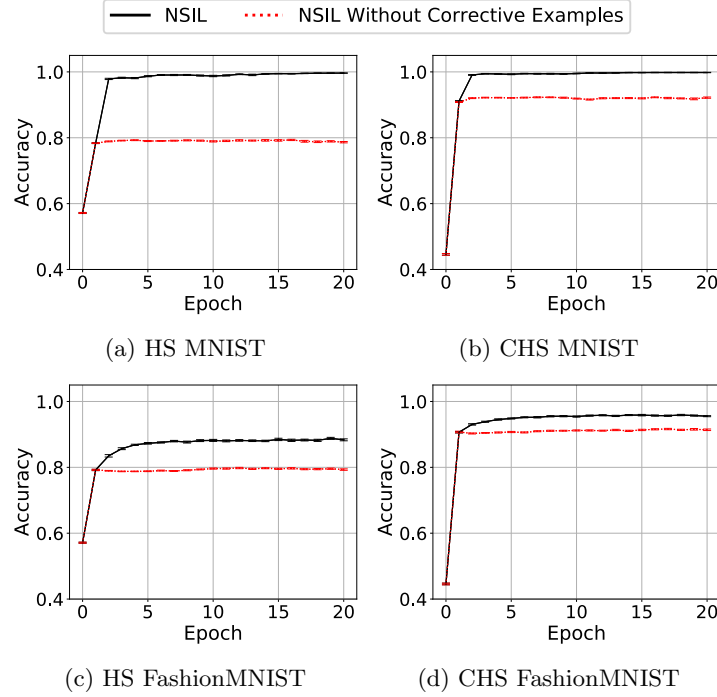


Figure 4.18: Hitting Set accuracy comparison with no corrective examples. Average over 20 repeats.

NSIL without corrective examples, in order to demonstrate the impact of the novel contribution of this chapter. Recall that we study the first two ablations by evaluating NeurASP (Yang et al., 2020), which assumes the symbolic knowledge is given, and FF-NSL (Cunnington et al., 2023a), which pre-trains the neural network on a separate dataset with concept label information. For the first two ablations, we evaluate the mean performance and standard error over 5 repeats. The third ablation is then evaluated by comparing NSIL with and without corrective examples. The results for the first two ablations are presented in Table 4.8. The results indicate the final task accuracy after performing a forward pass on the neural and symbolic components. In all task variants, NSIL achieves comparable accuracy to the fully supervised approach of FF-NSL, even when *no concept labels* are observed during training. Also, NSIL achieves comparable accuracy to NeurASP, whilst also *learning* the symbolic knowledge.

For the third ablation study, we compare the main NSIL results from Figure 4.16 to NSIL without corrective examples. The results are averaged over 20 repeats, and are presented in Figure 4.18. Since the correct hypothesis is not learned on the first iteration, the corrective examples enable NSIL to converge to a more accurate hypothesis. This clearly demonstrates the effectiveness of our proposed method. Finally, we present the accuracy against the learning time, for 20 training epochs, in Figure 4.19. As observed in all task variants, NSIL requires more time to complete 20 epochs, but ultimately achieves a greater accuracy than the neural baselines.

4.3.5 Investigating the λ Hyper-Parameter

We now investigate the λ hyper-parameter for NSIL introduced in Section 4.2.3, and its effect on the optimisation of the symbolic learner. When NSIL calculates the weights of the corrective examples, two updates occur; one based on the $\text{FNR}_{H',\theta'}(M, y)$, and the other based on $\text{CONF}_{\theta'}(Z, M, y)$. λ can be adjusted to vary the effect of the $\text{CONF}_{\theta'}(Z, M, y)$ update in

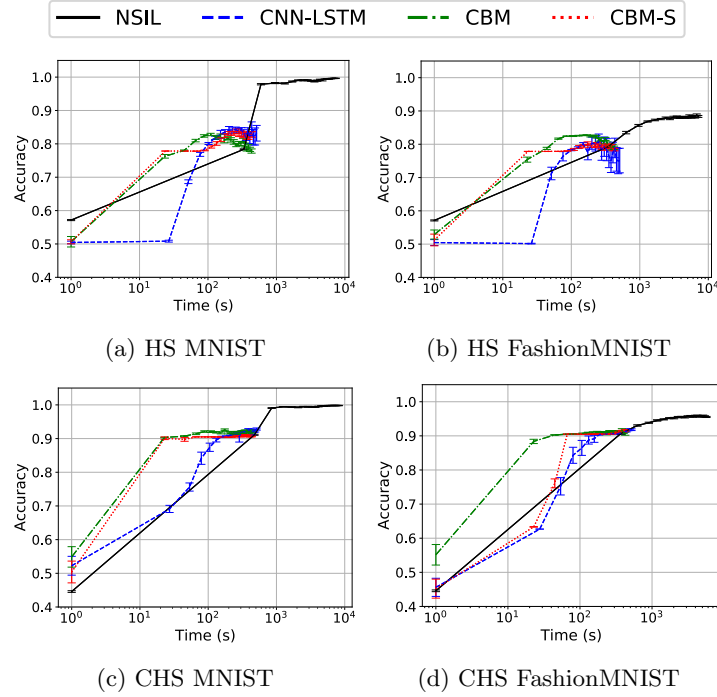


Figure 4.19: Hitting Set learning time vs. accuracy.

Equations 4.13 and 4.14, as the user may not want to fully rely on the neural network confidence when the neural network is under-trained. Therefore, we explore different values of $\lambda \in \{1, 0.8, 0.6, 0.4, 0.2, 0\}$, and record the accuracy of the learned hypothesis at each iteration. We select the more challenging E9P and HS tasks, and use 40% of the training data in the E9P task, to obtain results in a timely manner. We run NSIL for 10 iterations, with 5 repeats, using different randomly generated seeds. Figure 4.20 presents the results.

In all tasks, setting $\lambda = 1$ results in the best performance, as NSIL converges to the correct hypothesis. This is our justification for using $\lambda = 1$ in Section 4.2.3. In the E9P task, when $\lambda = 0$ (purple dash-dot line), the accuracy improves throughout training and the symbolic learner converges to the correct hypothesis at iteration 7, although has to explore more of the search space to do so. When $0 < \lambda < 1$, accurate hypotheses are learned until iteration 4, at which point the accuracy begins to fluctuate. In these cases, the corrective examples with accurate neural network predictions have lower weights, which encourages the symbolic learner to choose a different label than the label given in the training set. When $\lambda = 1$, NSIL maintains the correct hypothesis in later stages of training. In both HS tasks, $0 < \lambda \leq 1$ results in similar performance, whereas when $\lambda = 0$, the learned hypothesis alternates at each iteration. This is due to the “broad-brush” nature of the FNR in these tasks, as many examples have the same weight and are updated simultaneously. There are only two possible labels for each subset configuration, and there are many choices of digits, so the FNR for each label and subset configuration applies to many corrective examples. Note this is not the case in the E9P task, as there are 19 labels and fewer choices of digits for each label. In the HS tasks, when $\lambda > 0$, the neural network confidence scores help the symbolic learner to converge to the correct hypothesis by providing a positive weight update for specific examples. This breaks down the broad weights set by the FNR.

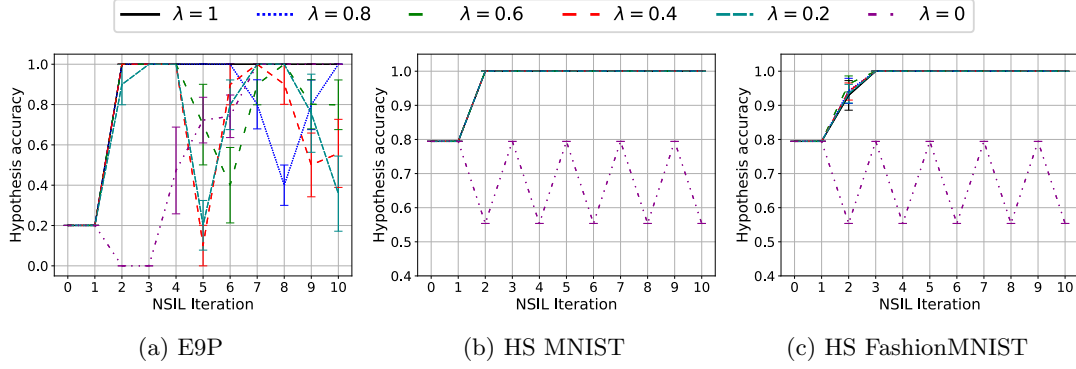


Figure 4.20: Hypothesis accuracy with varying λ , E9P and HS tasks. Error bars indicate standard error.

4.4 Challenges

Whilst we have demonstrated the effectiveness of NSIL on a variety of learning tasks, there are a few challenges to overcome if NSIL is to scale to more complex problems. In this section, we outline these challenges, and where possible, present alternative solutions. This helps to motivate the work in the next chapter.

4.4.1 Trying to Prove Convergence

It is interesting to consider whether we can prove any theoretical properties of the NSIL framework, in order to guarantee convergence to an optimal hypothesis, since the LAS symbolic learners are guaranteed to learn an optimal hypothesis with respect to a symbolic learning task given as input. If we could guarantee convergence, then it would be possible to define a termination condition, to avoid having to select the number of iterations to train NSIL for. Let us assume H_i and H_j are hypotheses learned at subsequent iterations of the NSIL algorithm. Let us also assume N_i and N_j are neural networks trained using H_i and H_j respectively. To simplify the problem, let us also assume that for a given hypothesis H , we always obtain the same neural network N . Whilst in reality, the neural network weights persist from one iteration to the next (and thus, one could obtain slightly different neural networks given the same H), we observe that given enough data, a neural network trained with the same hypothesis results in the same set of corrective example weights for the next iteration. Finally, we also assume that the neural networks are configured with the optimal hyper-parameters. In order to prove convergence, we attempt to find a counter example to it, in the form of a loop, where:

$$H_i \rightarrow N_i \rightarrow H_j \rightarrow N_j \rightarrow H_i \rightarrow N_i \quad (4.15)$$

If no such loop exists, then we could guarantee convergence to an optimal hypothesis. To assist us in the proof, let us first introduce some additional notation. We assume:

- H_i is learned from the set of examples E_i .
- H_j is learned from the set of examples E_j .
- E_j has weights calculated using H_i and N_i .
- E_i has weights calculated using H_j and N_j .

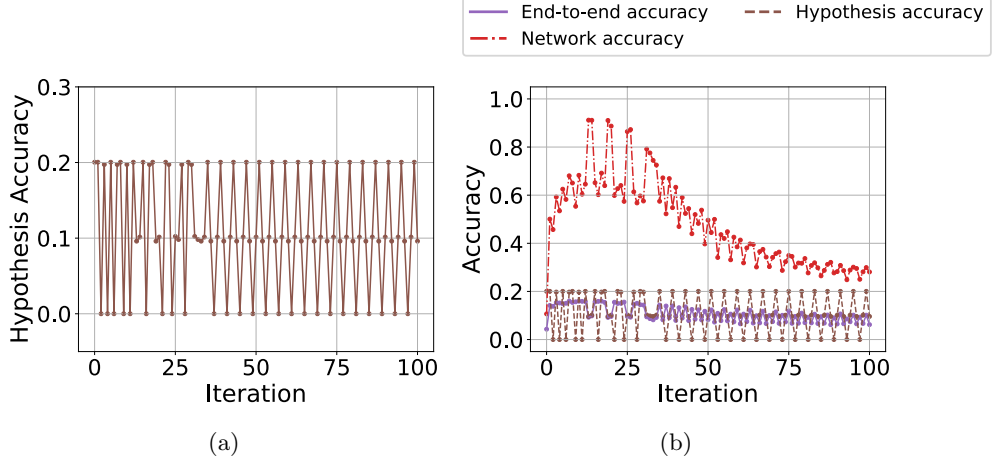


Figure 4.21: NSIL performance in a restricted search space for the Two-Digit Addition task. (a) The accuracy of the learned hypothesis only. (b) The accuracy of the neural network, the learned hypothesis, and the overall (end-to-end) accuracy. This shows NSIL is not guaranteed to converge to an optimal hypothesis.

- $PEN(H_i, E_i)$ is the total weight penalty of the uncovered examples in E_i , given H_i .
- $ACC(H_i, N_i)$ is the end-to-end task accuracy, given H_i and N_i .

We now run the Two-Digit Addition task from Section 4.3.3, but replace the addition mode body declaration with a product operator. This ensures it is impossible to learn the correct hypothesis (since the addition operator is removed from the hypothesis search space), and enables us to investigate convergence to a sub-optimal hypothesis. We run the task for 100 iterations, analyse all combinations $\langle H_i, N_i \rangle$, $\langle H_j, N_j \rangle$, where $i, j \in \{1 \dots 100\}$, and $i \neq j$, and test for the following conditions:

1. $PEN(H_j, E_j) < PEN(H_i, E_j)$
2. $PEN(H_i, E_i) < PEN(H_j, E_i)$
3. $ACC(H_i, N_i) > ACC(H_i, N_j)$
4. $ACC(H_j, N_j) > ACC(H_j, N_i)$

If 1 is true, then H_j is always learned after H_i , because the penalty of the uncovered examples is lower when calculated using H_j , than H_i . If 2 is true, then H_i is always learned after H_j , because the penalty of the uncovered examples is lower when calculated using H_i , than H_j . If 3 is true, then N_i will always be obtained over N_j , when trained with H_i . Finally, if 4 is true, then N_j will always be obtained over N_i , when trained with H_j . If we can find an i, j combination that satisfies all four conditions, then we have found a counter example, and proven NSIL is *not* guaranteed to converge to an optimal hypothesis.

When running this analysis, we find at iterations $i = 1$ and $j = 2$, hypothesis and neural network pairs that satisfy the four conditions. We restrict the hypothesis search space to only include H_1 and H_2 , and we observe an infinite loop: $H_1 \rightarrow N_1 \rightarrow H_2 \rightarrow N_2 \rightarrow H_1 \rightarrow N_1$. Furthermore, when running with the full search space (although with the addition operator replaced with the product operator, i.e., the correct hypothesis is removed), we also observe an infinite loop, except over a repeating sequence of four hypothesis and neural network pairs. This is shown by plotting the accuracy of the learned hypothesis at each iteration, shown in

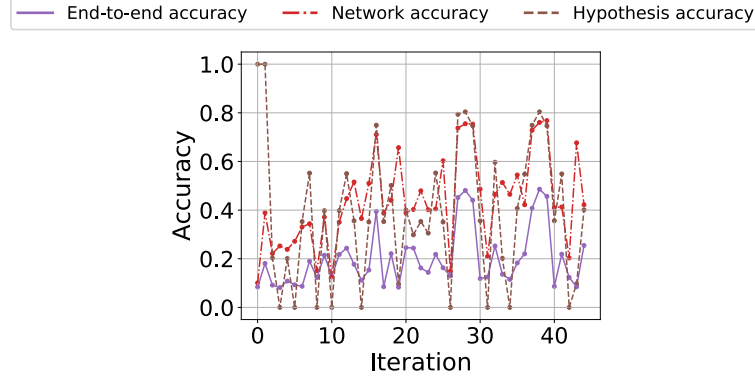


Figure 4.22: NSIL performance on the Two-Digit Addition task with CIFAR-10 images.

Figure 4.21a. Therefore, there is no guarantee NSIL will converge to the optimal hypothesis. Looking at Figure 4.21b, it is interesting to note that NSIL is able to train the neural network to a reasonable level of accuracy in the early iterations. Had the correct hypothesis existed in the search space, the neural network accuracy would have enabled NSIL to find the correct hypothesis via the positive corrective example weights. However, since this wasn't the case, the neural network accuracy eventually decreased as it was continuously trained via an incorrect hypothesis.

4.4.2 NeurASP

NSIL relies upon NeurASP to perform neuro-symbolic reasoning and train the neural network, given a candidate hypothesis. NeurASP has a few limitations. First, the NeurASP implementation (Yang et al., 2020) assumes a batch size of 1. This makes it challenging to train more complex perception components that rely upon a larger batch size. For example, (Aspis et al., 2022) have demonstrated training NeurASP on the Two-Digit Addition task, where the MNIST images are replaced with CIFAR-10 images (Krizhevsky et al., 2009), which is a significantly harder perception task. They use a ResNet56 architecture (He et al., 2016), and only achieve 6.52% task accuracy. Also, NeurASP only completed 1 epoch of training within their 4 hour time limit. When attempting this task in NSIL, with a simpler VGG16 network (Simonyan and Zisserman, 2015), we also obtain poor results, shown in Figure 4.22. In this case, NSIL struggles to achieve a high accuracy, and fails to converge to the correct hypothesis, despite learning the correct hypothesis on the first iteration.

Whilst it may be possible to engineer NeurASP to accept a larger batch size, the main limitation is that NeurASP enumerates all the answer sets of the ASP program $\Pi_{\langle M, y \rangle}$. This makes it difficult to scale to tasks with a greater amount of choice in the possible values for the concepts, as the gradients that are calculated for training the neural network are shared between a greater number of possibilities. For example, (Aspis et al., 2022) have demonstrated NeurASP fails to solve the MNIST Addition tasks, when an increasing number of digits are used in each data point. Therefore, a natural question to ask is whether we can swap NeurASP for an alternative, more scalable neuro-symbolic reasoning approach, to scale NSIL to more complex tasks. One possible candidate is Embed2Sym (Aspis et al., 2022), which consists of two stages: (1) The neural perception component is trained in an end-to-end fashion via a MLP as the reasoning component. As this is a fully differentiable architecture, the only labels required are the downstream task labels. (Aspis et al., 2022) have shown that this is sufficient for the

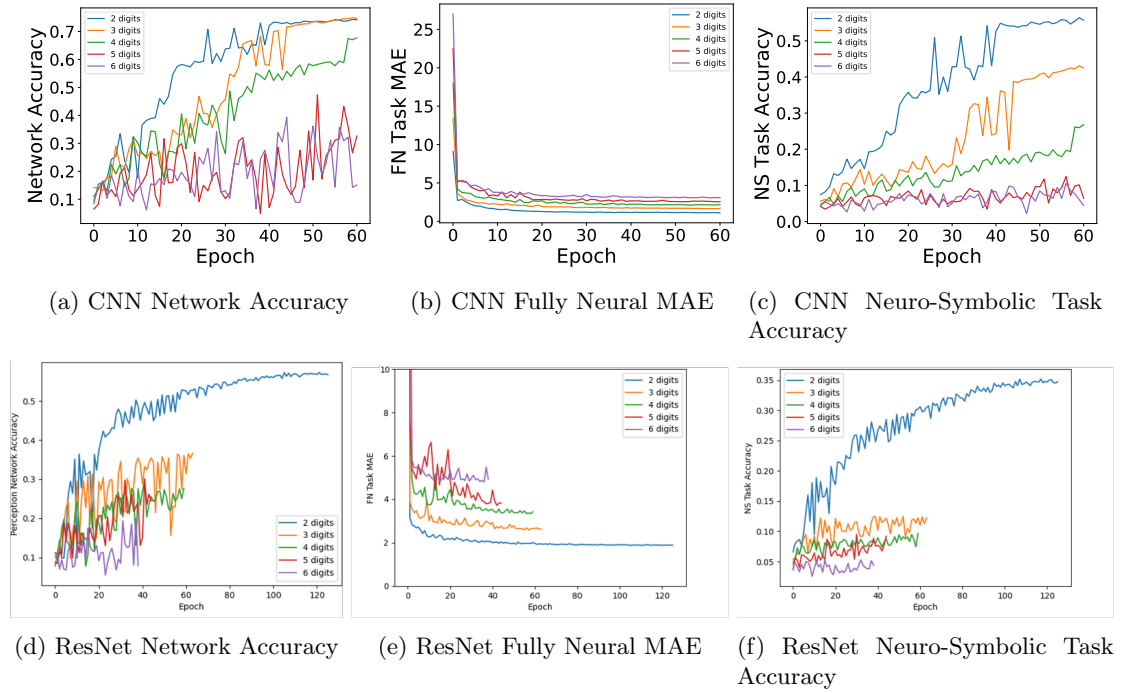


Figure 4.23: Embed2Sym performance on the CIFAR-10 Addition task with an increasing number of digits. (Top) a CNN is used as the perception component. (Bottom) a ResNet56 is used. (Left) The accuracy of the perception component on classifying concept values. (Middle) The MAE of the fully neural architecture from stage 1 of Embed2Sym (perception component + MLP). (Right) The final neuro-symbolic task accuracy from Embed2Sym.

perception network to cluster the concepts in an embedding space. (2) The MLP is discarded, and the embedding space is clustered using the K-means algorithm, before an ASP optimisation is used to find an optimal cluster labelling with respect to the downstream labels and the given hypothesis. Once trained, the cluster ID of a concept can be extracted from an unseen image using the perception component and the learned K-means clustering. Then, the cluster labelling from the ASP optimisation is used to map the cluster ID to the concept value. Together with the hypothesis, and (optionally) more concept values, the downstream label can be predicted. This technique is significantly more scalable than NeurASP, because it avoids having to enumerate all the possible answer sets, and, the perception network training in stage 1 can easily utilise a larger batch size.

However, we hypothesise that even the fully differentiable architecture of Embed2Sym is still difficult to scale through end-to-end training, especially when training the perception component in full. To investigate this, we train both a CNN, and a ResNet56 perception component with Embed2Sym on the CIFAR-10 Addition task, with an increasing number of digits. Figure 4.23 presents the results. It is clear that it becomes more difficult to train the neural network, and the entire architecture, as the number of digits increases. This is especially the case when training the ResNet (Figures 4.23d — 4.23f). In these cases, we had to implement a 24-hour timeout, which was reached as soon as we increased the number of digits to 3. Inspecting the embedding space of the perception components in Figure 4.24, we can also see that it becomes more difficult for the perception network to cluster the concepts, as the number of digits increases. For example, comparing the embedding space plots at Epoch 60 in Figure 4.24a, 10 distinct clusters are visible in the 2-digit case, whereas the clusters are less defined in the 6-digit case.

The ASP optimisation stage of Embed2Sym may also be difficult to scale to more complex

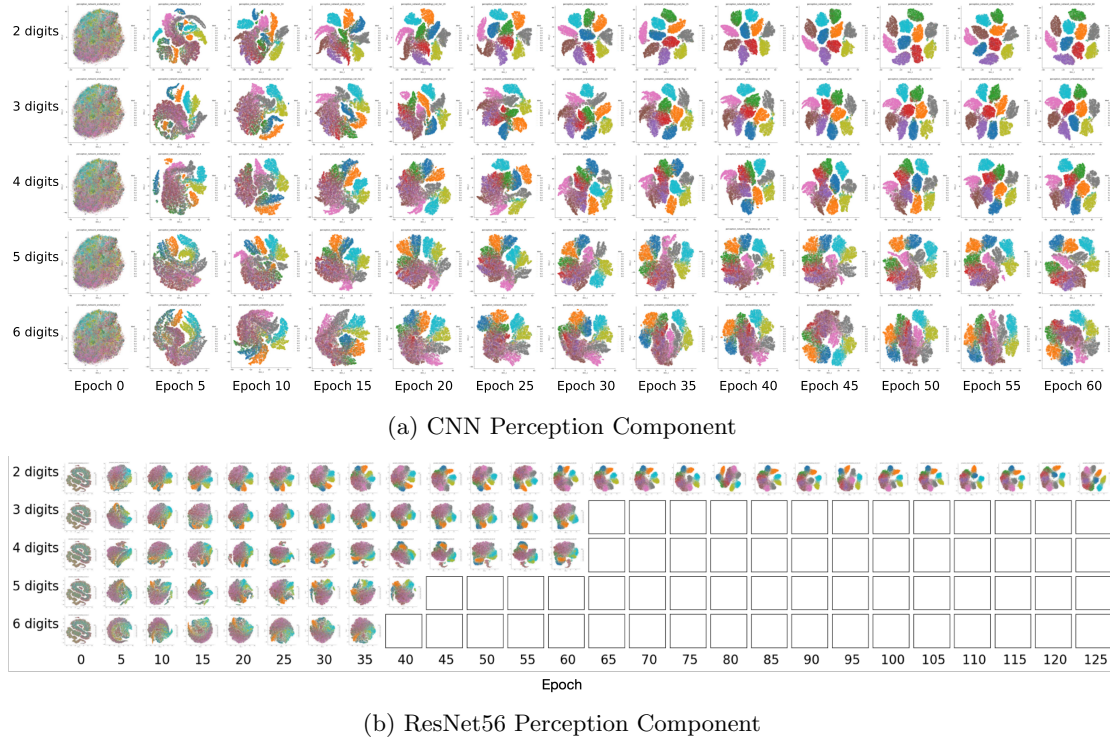


Figure 4.24: The embedding spaces of the perception components within Embed2Sym during training, as the number of digits increases.

tasks. We attempted cluster labelling when given perfect clusters (synthetically generated) for a Playing Card Addition task. This is similar to the MNIST Addition task, except the digits are represented by playing cards and can therefore take the values 0–51, instead of 0–9. Given two cards (i.e., an addition of two numbers with values 0–51), Embed2Sym was able to successfully label the clusters, given the ground-truth addition hypothesis, and 100 examples with downstream labels, indicating the sum of the two digits. We also obtained a successful cluster labelling with 3 cards, although we had to modify and experiment with the hyper-parameters in the Clingo ASP solver. When increasing the number of cards to 4, Clingo became stuck in the grounding stage, before solving occurred. The assumption for perfect clustering is unrealistic, and we hypothesise even with small amounts of noise, this is likely to have a significant impact on the run-time of the ASP optimisation.

This clearly illustrates the challenge of training a perception component in full, in an end-to-end manner via a reasoning task, given our analysis of the state-of-the-art neuro-symbolic reasoning approach Embed2Sym. Therefore, swapping NeurASP for Embed2Sym in NSIL is unlikely to yield any significant long-term benefit. Also, NSIL itself suffers a similar limitation to NeurASP, when calculating the corrective example weights based on the FNR, as this also depends on calculating the answer sets of $\Pi_{\langle M, y \rangle}$. Finally, NSIL relies on a bootstrap symbolic learning task that also enumerates the full choice of concept values. As a result, there are numerous challenges involved in scaling NSIL to harder problems.

4.5 Summary

In this chapter, we have introduced a novel method of performing iterative neuro-symbolic learning, called NSIL. NSIL has achieved state-of-the-art results on various neuro-symbolic bench-

marks, outperforming $Meta_{Abd}$, and a variety of deep learning baselines. This is the first approach to train a general neural network architecture, and learn complex and expressive ASP programs, in an end-to-end fashion. Crucially, NSIL does not require labels for the values of concepts, which addresses the main limitation with our previous FF-NSL approach in Chapter 3. The novelty in NSIL, is the method of biasing the symbolic learner using corrective examples, weighted to balance exploration and exploitation of the hypothesis search space. Our evaluation has demonstrated the corrective examples enable NSIL to converge to the correct hypothesis in large search spaces, even when the neural networks are trained with limited data.

However, we have also identified numerous challenges with this approach, if the architecture were to be scaled up and applied to more complex problems. First, we have proven NSIL is not guaranteed to converge to the optimal hypothesis, and may become stuck in an infinite loop, if the correct hypothesis does not appear in the search space. Also, as NSIL relies upon a neuro-symbolic reasoning approach to train the neural network, this also limits the scalability due to enumerating the full space of the concept values. Finally, as NSIL adopts an iterative training approach with two components, the learning time is slow compared to neural baselines, although ultimately, accurate, and interpretable knowledge is learned given enough time.

In the next chapter, we pursue a scalable neuro-symbolic reasoning approach, with the goal of developing a drop-in replacement for NeurASP in the NSIL framework. We are specifically interested in whether we can utilise advancements in unsupervised, and self-supervised learning, to train a perception component without concept labels. Whilst NSIL has solved the drawback of FF-NSL, by eliminating the requirement to label the concept values, this has come at the expense of scalability. We therefore proceed with the goal of combining the benefits of these two approaches, whilst eliminating both drawbacks.

Chapter 5

Exploring Decoupled Neuro-Symbolic Learning and Reasoning

As discussed in Chapter 4, our NSIL framework relies upon a neuro-symbolic reasoning approach to train the neural network, given a candidate hypothesis. We have demonstrated that current neuro-symbolic reasoning approaches lack scalability, when the reasoning task is directly coupled with the end-to-end training of a perception component. In cases where there is a greater number of choices for the values of the primitive concepts, we hypothesise that the gradient signal becomes weak, as it is shared between many possibilities. This could explain why our experiments show it becomes more difficult to train larger perception architectures that consist of many layers (e.g., a ResNet), as the number of concept values increases. Therefore, in this chapter, we explore *decoupling* the training of the perception component from the reasoning task, with the goal of developing a scalable neuro-symbolic reasoning approach that can be used as a plug-in replacement for NeurASP in our NSIL framework. The current state-of-the-art neuro-symbolic reasoning approach in terms of scalability is Embed2Sym (Aspis et al., 2022). Recall Embed2Sym consists of two stages: (1) A clustered embedding space is learned by training the perception component end-to-end via a fully differentiable reasoning network. (2) An ASP optimisation is used to label the clusters with concept values, given a symbolic hypothesis representing the rules of a reasoning task, and a dataset of cluster predictions output from stage 1, annotated with downstream labels. Inspired by Embed2Sym, in this chapter we aim to develop more scalable alternatives of these two steps.

To learn a clustered embedding space without concept labels, we consider a variety of approaches, including unsupervised learning (Kingma and Welling, 2013), self-supervised learning (Jaiswal et al., 2020), and using a large pre-trained model as a feature extractor. These approaches enable the decoupling of the perception network from the reasoning task during training, as they can be trained independently without any labels or downstream supervision. Once the networks are trained, we plan to inspect the embedding space, in a similar manner to Embed2Sym, to determine whether a clustering of concept values can be obtained. Note the decoupling ensures the bottleneck regarding the choice of concept values is eliminated, for training the perception network. For example, if a self-supervised approach is able to cluster MNIST digit images, and assuming 1) the rules of the reasoning task are given, and 2) the clusters can be labelled correctly, the number of digits in an MNIST Addition task could be scaled without

impacting the image classification performance for single digit images. The task accuracy would degrade gracefully according to the theoretical limit a^n , where a is the accuracy of classifying single digit images, and n is the number of digits in the Addition data point. Contrast this to the current Embed2Sym approach that trains the perception component end-to-end. As we demonstrated in Chapter 4, the accuracy of the perception component decreases as the number of digits increases, which we hypothesise is due to a weakening gradient signal during end-to-end training (see Figure 4.23).

To label the clusters, we introduce two novel approaches that map cluster identifiers to concept values. The first approach is differentiable, since differentiable methods tend to be the easiest to scale. The second approach utilises a breadth-first search to label the clusters, and attempts to simultaneously *learn* a hypothesis using a symbolic learner. In addition to labelling the clusters, the idea is to solve the optimality guarantee problem with NSIL, as even with a scalable neuro-symbolic reasoning approach, learning an optimal hypothesis remains an outstanding challenge. The remainder of this chapter is organised as follows. In Section 5.1, we present a set of experiments evaluating the various unsupervised, and self-supervised clustering approaches on MNIST, CIFAR-10, and playing card images. In Section 5.2, we present our two novel methods of labelling the clusters, alongside the experimental results. Finally, in Section 5.3, we summarise our findings, and outline the direction for our final contribution of this thesis.

5.1 Image Clustering

In this section, we briefly evaluate unsupervised learning, self-supervised learning, and a pre-trained feature extractor, and investigate whether we can achieve concept clustering within an embedding space, without using concept labels, and crucially, without end-to-end training. Let us now present our analysis, including a brief overview of the various techniques studied.

Unsupervised Learning Deep learning without labels requires a modification to the architectures introduced in Chapter 2. Many unsupervised deep learning approaches are based upon the *autoencoder*. The general idea is to learn a compressed latent representation of the input data, which can be used to output a reproduction of the same input data. This is usually achieved by using two sub-networks with the same “mirrored” architecture, called an *encoder* and a *decoder*. The encoder maps the high-dimensional raw input data to the compressed latent representation, and the decoder performs the opposite. The encoder and decoder can in principle be of any neural architecture.

To train such an architecture, it is sufficient to simply treat the input data as the target label, with the objective of minimising the distance between the predicted output, and the original input. A popular loss function is the mean squared error function, which minimises the squared distance between the predicted and target values, summed across a set of training data points (see Equation 2.10 in Chapter 2). In unsupervised learning, this loss is known as the *reconstruction loss*, since it measures the error in reconstructing the original input. At this point, a natural question to ask is why such an architecture is useful, if once trained, the network can only reproduce a given piece of input data? In order to minimise the reconstruction loss across a sufficiently large training set, the network is forced to learn a latent representation that models the key features in the input data. For example, if the network is trained to reproduce MNIST images, the latent representation must represent an image of a digit 2 differently than digit 3, otherwise the network would mistakenly output the same image for different inputs. This

is useful in our case, as it is possible to inspect the learned latent representation and perform clustering.

A popular extension to the autoencoder is the Variational Autoencoder (VAE) (Kingma and Welling, 2013), which adds a probabilistic component to the architecture, as well as an additional term to the loss function. Instead of the encoder outputting a fixed point in the latent space, the encoder is trained to output the mean and variance parameters of a probability distribution. With this approach, a latent vector can be sampled from this distribution, to generate novel outputs. This is useful in generative tasks such as image synthesis, where data points can be generated that do not already exist in the training data. Not only is the network trained to minimise the reconstruction loss, it is also trained to learn a latent distribution that follows a Gaussian distribution, to ensure a smooth continuous structure. A smooth structure enables easier interpolation between data points to generate new data points of interest. This is achieved by minimising the Kullback-Leibler divergence (Kullback and Leibler, 1951) between the current latent distribution, and a reference Gaussian distribution.

Given the popularity of the VAE, we evaluate it when applied to the simplest of our image datasets, the MNIST dataset, with the goal of obtaining a clustered latent representation. Our method is as follows. The VAE architecture consists of a MLP encoder and decoder model, where the MLP has 3 linear layers with the ReLU activation function attached to the first layer. The first linear layer encodes the features of the images, whilst the second and third linear layers are tasked with outputting the mean and variance parameters of the latent distribution. Once the network is trained, we inspect the latent representation, to identify whether the network has learned to cluster the MNIST digits into 10 distinct clusters. Therefore, we restrict the size of the latent distribution to 2 dimensions, for visualisation. We train for 20 epochs with a batch size of 128 on the full MNIST training set.

Figure 5.1 shows the learned latent representation, alongside the reconstruction loss. It is clear that the network has learned to cluster the images somewhat, although there aren't any *distinct* clusters, which presents a challenge for a clustering algorithm such as K-means. Furthermore, the reconstruction loss has converged, and decreased significantly from the original value before training. This suggests that the network has learned to solve the task of reproducing the input images, without learning a latent representation that is amenable to clustering. The problem is that there are no constraints during training to explicitly encourage the network to form a *clustered* latent space, it can perfectly minimise the reconstruction loss by simply encoding each image into a different part of the latent space. Therefore, we now turn our attention to self-supervised approaches, which extend ideas from unsupervised learning to include more explicit clustering objectives during training.

Self-Supervised Learning Broadly speaking, self-supervised learning involves the generation of pseudo-labels that can be used during training. Crucially, these are not human provided, they are generated internally within the system architecture. There are a variety of approaches to achieve this. Some extend unsupervised architectures with a supervisory signal applied to generated data points. For example, (Mahon and Lukasiewicz, 2021) achieve 99.2% clustering accuracy¹ on the MNIST dataset, by pre-training an ensemble of autoencoders, performing clustering on the learned representations using k-means, and then identifying a set of agreed cluster predictions across the ensemble to generate a supervisory training signal. The results

¹An accuracy metric can be computed by first running a K-means algorithm to generate cluster assignments for each image. For the purposes of scoring, the cluster assignments can be mapped to the ground-truth labels, according to a majority vote, so it is possible to determine if each image has been predicted correctly. This can be achieved using the Hungarian matching algorithm (Kuhn, 1955).

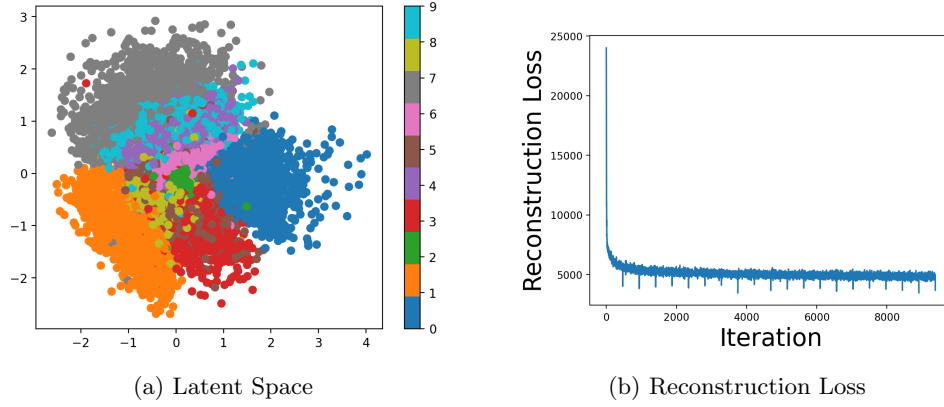


Figure 5.1: Training a Variational Autoencoder on MNIST images.

in (Mahon and Lukasiewicz, 2021) indicate their approach outperforms all other unsupervised techniques on the MNIST clustering task. With such a high accuracy, we can consider clustering over the MNIST dataset without concept labels a solved problem. However, (Mahon and Lukasiewicz, 2021) only achieve 67.9% accuracy on the FashionMNIST dataset, which is a harder perception problem, but one that can be solved to an accuracy of 96.91% with a fully supervised approach (Tanveer et al., 2021). Therefore, we proceed to evaluate other self-supervised techniques that can be applied to more complex imagery.

Many self-supervised techniques replace the reconstruction loss of an autoencoder with a *contrastive loss*, which is more aligned with the clustering objective (Hadsell et al., 2006; He et al., 2020; Misra and Maaten, 2020; Chen et al., 2020). The idea is to use *siamese encoders*, (i.e., two copies of the same encoder network with shared weights), that encode the raw input data into an embedding space. Instead of aiming to reproduce the original input using a decoder, one encoder receives the original raw input, and the other encoder receives a perturbed version of the same input, which creates a *positive* training pair. It is also possible to create a *negative* training pair, by passing an entirely different input to the second encoder. The training objective is to ensure positive pairs have a small cosine distance in the embedding space, and negative pairs have a large cosine distance. As these approaches train *without* concept labels, positive pairs are often generated through an image transformation (e.g., a brightness or colour shift) of a single raw input, and negative pairs are sampled randomly from the dataset, or generated synthetically. This provides an improved training objective compared to the reconstruction loss, because the network is explicitly encouraged to separate the input data in the embedding space, rather than just reconstructing the input data as output. The contrastive loss prevents the network from learning a “look-up table” that effectively memorises the training data, which could occur with the reconstruction loss.

To evaluate a self-supervised approach, we analyse the SPICE technique (Niu et al., 2022), since SPICE has achieved a state of the art clustering accuracy of 91.8% on the challenging CIFAR-10 dataset. SPICE consists of three stages: (1) A ResNet feature extractor is trained via the contrastive loss, to enforce transformations of the same image to have similar representations in the latent space, and different images to have distinctly different representations. (2) The weights of the feature extractor are frozen, and a clustering head is trained to map feature vectors to a probability distribution over the expected clusters. The clustering head is simply a linear layer. Since no ground-truth labels are available, an expectation-maximisation step generates pseudo-labels for training, in a similar fashion to the K-means algorithm. (3) The clustering

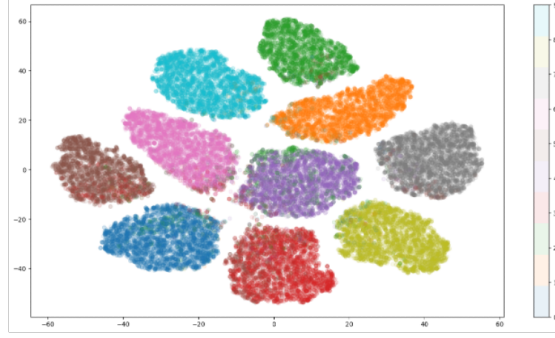


Figure 5.2: Evaluating the SPICE clustering method on CIFAR-10 test images.

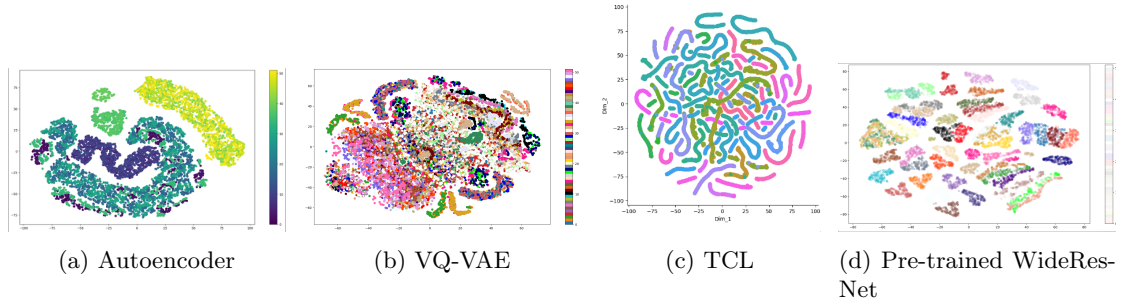


Figure 5.3: Playing card clustering with full card images.

head and the feature extractor are trained end-to-end using the pseudo-labels generated from step 2 that have high confidence, where confidence is defined as the distance of the sample point in the latent space to the cluster centroid.

Since the authors of SPICE have open-sourced their model weights, we can simply evaluate their trained clustering model for CIFAR-10 images, out-of-the-box. We generate a T-SNE plot, which projects the probability vector output from the clustering head into 2 dimensions, for each image in the CIFAR-10 test set. The results are shown in Figure 5.2. As you can see, the SPICE model is able to separate the CIFAR-10 images into distinct clusters. Since SPICE achieves such high accuracy, we consider the CIFAR-10 dataset to be a solved problem in terms of clustering.² Recall from Chapter 4 that the first stage of Embed2Sym, which trains a ResNet end-to-end to cluster CIFAR-10 images via a reasoning task, is challenging to scale when an Addition reasoning task is used with > 2 digits, and the CIFAR-10 images are mapped to digit values 0–9 (see Figure 4.24). We now turn our attention to the more difficult playing cards dataset, which has 52 distinct clusters, instead of 10 for MNIST and CIFAR-10.

Clustering the playing cards dataset Clustering the playing cards is a much harder problem, given the increased number of clusters, but also due to the visual similarity of the images (e.g., the card images for the numeric ranks of hearts and diamonds are visually very similar). Unfortunately, we cannot evaluate the previously studied SPICE method on this dataset, since this requires training our own SPICE model. The SPICE authors indicate that 4 GPUs are required to train SPICE on CIFAR-10.³ We attempted to train SPICE with 1 GPU, but failed to obtain satisfactory results within a reasonable time-frame.

Therefore, we conduct a broader analysis of a variety of existing approaches, both unsuper-

²Since this analysis was undertaken, the current best performing approach for CIFAR-10 achieves 99.5% clustering accuracy (Gadetsky et al., 2024).

³<https://github.com/niuchuangnn/SPICE/blob/main/train.md>

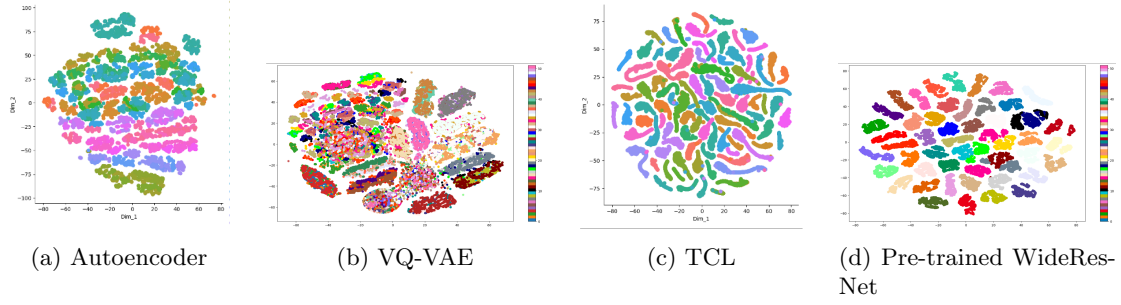


Figure 5.4: Playing card clustering with Rank Suit Crop images.

vised and self-supervised. We evaluate an autoencoder consisting of linear layers within the encode and decoder networks, a state-of-the-art variational autoencoder called VQ-VAE (Van Den Oord et al., 2017), a self-supervised approach called Twin Contrastive Learning (TCL) (Li et al., 2022c) with the best feature extractor available, a WideResNet. Notably, TCL has outperformed SPICE on a variety of tasks, namely, ImageNet-10 which is a harder dataset than CIFAR-10. Finally, we evaluate an off-the-shelf WideResNet feature extractor which is pre-trained on ImageNet. We train the autoencoder for 60 epochs, with a batch size of 256, a latent embedding dimension of 512, a learning rate of 0.001, and the Adam optimizer with a weight decay of 0.00001. On both datasets the reconstruction loss approaches 0.001, indicating the network has converged. For the VQ-VAE, and TCL approaches, we use the default configuration given in the respective papers. In all methods, we extract the learned clustered latent space and visualise it using T-SNE. For the WideResNet feature extractor, we simply run each image through the out-of-the-box architecture with a forward pass, and store the resulting embedding for plotting with T-SNE. Since there is no specific fine-tuning for these datasets, there is no training configuration to report. All reported cluster accuracy metrics are obtained by performing a K-means clustering on the learned embedding space, followed by the Hungarian matching algorithm. The Hungarian matching algorithm performs a majority vote by first identifying for each ground-truth label which cluster prediction is most commonly predicted, and then assuming the ground-truth label as the final prediction whenever a cluster with this value is predicted.

We create an additional simplified playing cards dataset, called “Rank Suit Crop”, which contains cropped images of the top-left corner of each playing card, displaying the suit and rank symbol only. This is much easier than images of the full playing card, since there is no mirroring in the image (e.g., cards with the rank 6 do not include a mirrored 6 which looks like a 9), and the detail in the picture cards is removed. We use a training set of 39,000 examples and 27,300 examples with an even distribution across the 52 cards for the Rank Suit Crop and full playing cards datasets respectively. The resulting latent spaces for each method are presented in Figures 5.3, and 5.4. These figures show the latent embeddings for each test data point, where the full card dataset and the Rank Suit Crop dataset has 11,700 and 5,200 evenly distributed test data points respectively. Note that the simpler Rank Suit Crop dataset has a larger training set, in an attempt to improve the performance. This dataset was generated after inspecting the results on the full card images. As for the test dataset sizes, both datasets are sufficiently large enough to analyse whether a clustered embedding space can be learned, and the Rank Suit Crop dataset is smaller to provide an experimental speed-up.

First, all methods perform better on the simpler Rank Suit Crop images (Figure 5.4), than the full playing card images (Figure 5.3), which is unsurprising. However, all of the unsupervised and self-supervised approaches (Autoencoder, VQ-VAE, TCL) fail to separate the images

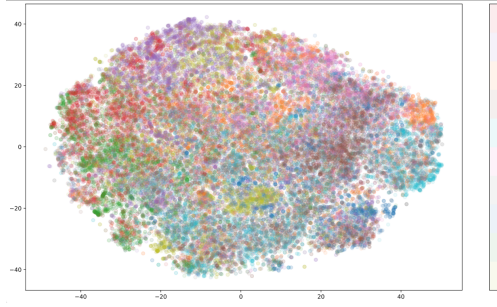


Figure 5.5: Evaluating the WideResNet feature extractor on CIFAR-10 test images.

into 52 distinct clusters on both datasets. The best performing model on both datasets is the WideResNet feature extractor. When running K-means and the Hungarian matching algorithm, the WideResNet achieves 68.7% and 92% accuracy, on the full playing card images and the Rank Suit Crop images, respectively. However, this result is suspicious, since no specific fine-tuning has occurred on this dataset. We hypothesise that features closely resembling playing cards have appeared in the ImageNet dataset which the WideResNet has been pre-trained on. To investigate this hypothesis, we evaluated the pre-trained WideResNet on CIFAR-10 images, for which to the best of our knowledge, there is no dataset contamination. In this case, the WideResNet completely failed to separate the images into distinct clusters, achieving 27% accuracy. The T-SNE plot is shown in Figure 5.5.

To summarise, our brief analysis of clustering methods has shown it is possible to achieve a reasonable level of accuracy without using concept labels, for the MNIST, CIFAR-10, and simple “Rank Suit Crop” playing cards dataset. However, there are also some challenges with these approaches when scaling to more complex problems. First, as shown in our analysis of the playing cards dataset, there is no guarantee the unsupervised and self-supervised learning approaches will yield a good clustering. We hypothesise that the research within these fields is largely focused around a similar benchmark of image datasets, and as the algorithms rely upon creating a set of specific image transformations to generate positive and negative data points, it is possible these transformations have been over-fit to the studied datasets. Although we did get a successful clustering with our WideResNet feature extractor on this dataset, this method only worked well on the simplified playing cards, and it also completely failed on CIFAR-10. Therefore, there currently does not appear to be a general method that succeeds on a wide range of datasets, which casts doubt over the validity of this direction. Nevertheless, this is an active field of research, and there may be such a method in future. Therefore, we proceed to develop a method of labelling the clusters, since the clusterings obtained in this section would still outperform those obtained using the state-of-the art Embed2Sym approach, assuming we can label the clusters correctly.

5.2 Labelling the Clusters

Recall our overall objective of this chapter is to develop a more scalable neuro-symbolic reasoning approach that can be used as a plug-in replacement for NeurASP within our NSIL architecture presented in Chapter 4. We are aiming to improve upon the results obtained by the best performing neuro-symbolic reasoning approach Embed2Sym, which has two stages: (1) Learning a clustered embedding space through end-to-end training, and (2) Labelling the clusters using an ASP optimisation. Achieving the first step without end-to-end training via a reasoning task

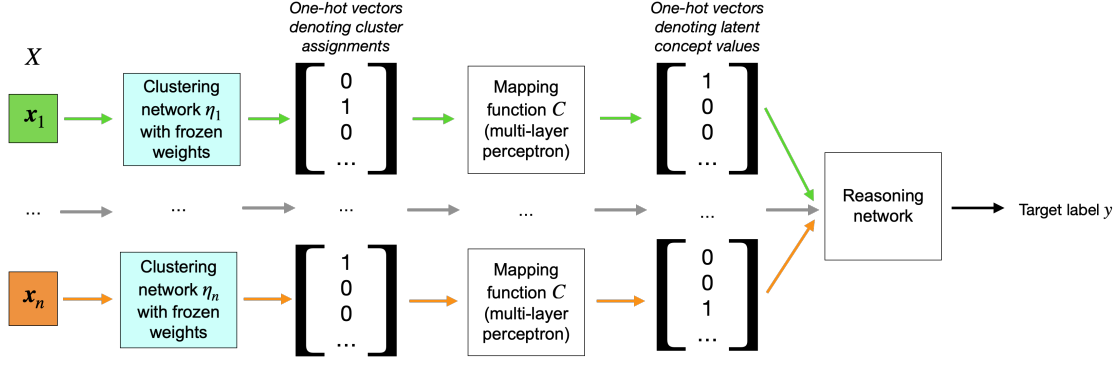


Figure 5.6: Fully differentiable end-to-end architecture for learning a cluster mapping function.

was addressed in the previous section, and in this section, we introduce two methods for labelling the clusters.

Let us assume we have a clustering method from Section 5.1, that trains a neural network η to map raw data to a probability distribution over k cluster assignments. As in previous chapters, there could be multiple neural networks η_i tasked with performing perception from a given raw input x_i . Without loss of generality, for the remainder of this chapter, we assume the raw input data is of a single type (e.g., all inputs are MNIST images), and therefore a single neural network is used. We refer to the neural network as a *clustering network*. To tackle the cluster labelling problem in isolation, let us also assume the clustering network returns a perfect one-hot vector over the cluster assignments. Note that in the case of uncertain assignments, the argmax function could be used to generate a one-hot vector, by setting the entry with the maximum probability to 1, and all other entries to 0. However, we focus on the ideal case of perfect clustering for the initial development of this component.

We present two novel directions in this section, a new differentiable approach for cluster labelling, and a purely symbolic approach, which can also learn a symbolic hypothesis. This aims to address the additional drawback of NSIL, namely, that it is not guaranteed to learn an optimal hypothesis with respect to the neural network predictions.

5.2.1 Fully Differentiable Cluster Labelling

In this approach, the goal is to learn a cluster mapping function $C : [0 \dots 1]^k \rightarrow [0 \dots 1]^k$ that maps a one-hot vector denoting the cluster assignment, to another one-hot vector denoting the value of the concept. We assume a symbolic hypothesis H is available, representing the rules of a reasoning task, along with a dataset $\mathcal{D}_{\text{task}}$ of labelled tuples of raw input data $\langle X, M, y \rangle$, where X is a sequence of images, M is the associated meta-data, and y is the target label. To learn C , one approach is to construct a fully differentiable end-to-end architecture in a similar fashion to Embed2Sym, as shown in Figure 5.6. In this architecture, the weights of the clustering network are frozen, and the mapping function C , as well as the reasoning network, are learned. C can be implemented using a simple MLP, since it is tasked with mapping a one-hot vector to another one-hot vector. Note C appears in both branches in Figure 5.6, but the weights are shared (i.e., it is the same network, it is just separated out for visualisation purposes). The reasoning network can be customised for the desired task. For example, it can be another MLP, or it can be a sequence model such as an LSTM for sequence classification tasks. Whilst it may be possible to train this network end-to-end, and thus learn both the mapping function C and the reasoning

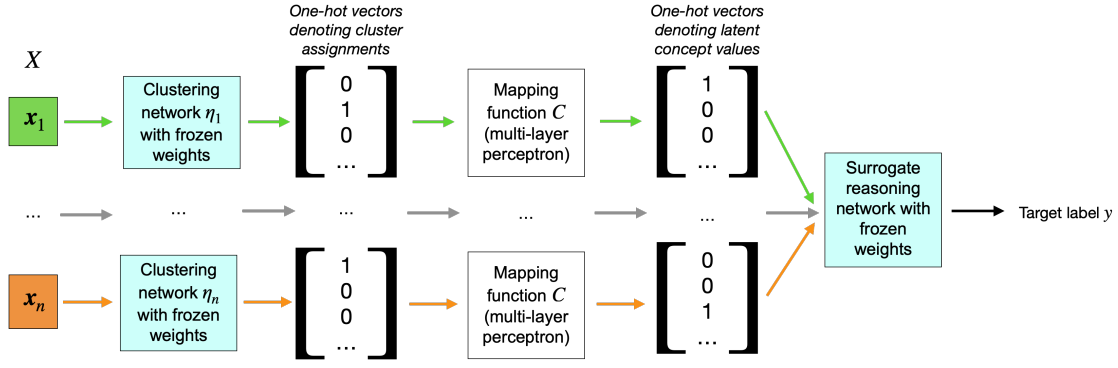


Figure 5.7: Fully differentiable end-to-end architecture for learning a cluster mapping function, where only the cluster mapping function is trainable.

network, we are not utilising the given hypothesis, which may provide vital information as to the behaviour of the mapping function. Without the hypothesis, there is nothing to prevent the network from learning an incorrect mapping function, that happens to perform well on the particular task. As the symbolic hypothesis is not differentiable, we cannot substitute the reasoning network with the hypothesis directly, but we can train a surrogate reasoning network to emulate the function of the hypothesis, and then use this surrogate network within the end-to-end architecture. This idea was also presented in the NeuroPlex framework (Xing et al., 2020), which trains a reasoning network using randomly generated training data. The data takes the form of one-hot input vectors representing a sequence of concepts, together with a downstream task label which is computed using the given hypothesis. Since the training data is generated randomly, a sufficiently large training set can be obtained without any human annotations.

Once trained, the differentiable reasoning network approximates the function of the hypothesis, so its weights can be frozen and the network can be re-used within an end-to-end architecture to train the perception component, as shown in Figure 5.7. Now, only the cluster mapping function is trainable, and it is constrained via the frozen clustering network, and the frozen reasoning network, which emulates the given hypothesis. This is a significantly easier task than training a perception component in full, such as a ResNet, within an end-to-end architecture as is performed in Embed2Sym. Now, it is possible to constrain our network architecture even further. Since the reasoning network expects one-hot vectors, the mapping function C should also output one-hot vectors. There is currently no constraint to enforce this within the architecture. Therefore, we add a Gumbel-Softmax layer (Jang et al., 2017) to the output of the mapping function. The Gumbel-Softmax is a differentiable categorical sampling function that can transform a standard probability distribution into a one-hot vector. Crucially, this operation is differentiable, so we can backpropagate gradients through it to train the mapping function. Now, there is one more constraint we can apply. The mapping function should ideally output a different concept value for each cluster assignment, i.e., it should behave like a permutation matrix. This property is currently not directly enforced. Therefore, we add an additional loss term to the mapping function, and combine it with the end-to-end training objective.

The intuition is to penalise the mapping function for predicting the same concept value for different cluster assignments. After each batch of end-to-end training, we input a batch of one-hot vectors, denoting every possible cluster assignment. For example, if there are $k = 4$ possible

concept values, we input a batch of four one-hot vectors, representing an identity matrix:

$$\begin{aligned} &[[1, 0, 0, 0], \\ &[0, 1, 0, 0], \\ &[0, 0, 1, 0], \\ &[0, 0, 0, 1]] \end{aligned}$$

The goal is for the mapping function to output a perfect permutation matrix, where there is exactly one entry of the value 1 in each row and column, and all other entries are equal to 0. To enforce this constraint during training, we calculate a loss which sums the values of the output vectors column-wise, and compares the result to the column-wise sum of the input identity matrix, using the MAE loss function. If the mapping function outputs a perfect permutation matrix, the loss will be 0, since the column-wise sum of both input and output will be equal, otherwise it will be greater than 0. We refer to this as the *permutation loss* $\mathcal{L}_P$, which is formally defined as:

$$\mathcal{L}_P = \text{MAE} \left(\left\langle \sum_{i=1}^k \hat{P}_{i,1}, \dots, \sum_{i=1}^k \hat{P}_{i,k} \right\rangle, \langle 1 \rangle^k \right) \quad (5.1)$$

where MAE denotes the mean absolute error function (see Equation 2.9), k is the number of concepts, and $\hat{P}_{i,j}$ represents the value at row i and column j of the predicted permutation matrix, as output by the mapping function for an input identity matrix. Note the first argument to the MAE function is the column-wise sum of $\hat{P}$, and the second argument is a vector of 1's with length equal to k (i.e., the column-wise sum of the identity matrix). As an example, if the mapping function outputs a perfect permutation matrix:

$$\begin{aligned} \hat{P} = &[[0, 1, 0, 0], \\ &[0, 0, 1, 0], \\ &[1, 0, 0, 0], \\ &[0, 0, 0, 1]] \end{aligned}$$

which indicates the first cluster is assigned to the second concept value, the second cluster is assigned to the third concept value, etc..., the column-wise sum of $\hat{P}$ is equal to $\langle 1, 1, 1, 1 \rangle$, which results in a permutation loss of 0. Alternatively, consider the case where the mapping function outputs an imperfect permutation matrix, and maps two different cluster assignments to the same concept value:

$$\begin{aligned} \hat{P} = &[[0, 1, 0, 0], \\ &[0, 1, 0, 0], \\ &[0, 0, 1, 0], \\ &[0, 0, 1, 0]] \end{aligned}$$

The column-wise sum of $\hat{P}$ is equal to $\langle 0, 2, 2, 0 \rangle$, which, when compared to the sum of the identity matrix columns using the MAE function, results in a permutation loss of 1. Assuming the standard end-to-end training objective is defined by the loss $\mathcal{L}_E$, the overall training objective is to minimise:

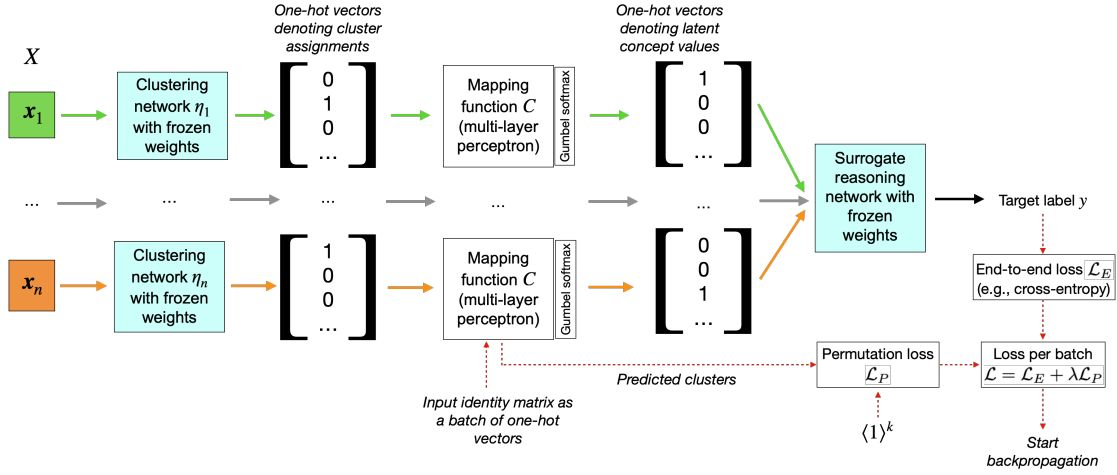


Figure 5.8: Fully differentiable end-to-end architecture for learning a cluster mapping function, with all of our modifications added.

$$\mathcal{L} = \mathcal{L}_E + \lambda \mathcal{L}_P \quad (5.2)$$

where λ is a hyper-parameter that can be used to balance the influence of the permutation loss against the end-to-end loss. Note the choice of end-to-end loss function is task dependant. For a classification task, a cross-entropy loss can be used, or for a numerical prediction task, the MAE can be used as a regression loss. Given these constraints, the final architecture is shown in Figure 5.8. This is a novel, and highly constrained architecture, capable of learning a mapping function from cluster assignments to concept values. Since we have decoupled the training of the perception component via clustering, we have effectively reduced the end-to-end training requirements to the most minimal architecture possible, and only the mapping MLP needs to be trained in an end-to-end fashion via a reasoning task. Let us now present our experimental results.

Experimental Results

Since our architecture in Figure 5.8 relies upon a frozen reasoning network, we first present the results of training this component with randomly generated one-hot vectors representing concept values. Our goal is to evaluate whether we can train a neural network surrogate to learn complex knowledge expressed by a symbolic hypothesis, especially knowledge that can be efficiently represented in ASP (i.e., using a small number of atoms). We choose the Hitting Sets tasks from Chapter 4, as these tasks require learning constraints and choice rules, the Recursive Arithmetic tasks from Chapter 4, as these task require learning recursive functions, and finally, the Follow Suit Winner task from Chapter 3, since this task requires predicate invention. In all tasks, the reasoning network is a 3-layer MLP, and we train for 100 epochs with a batch size of 8. In the Hitting Sets tasks, the MLP has a sigmoid output layer, since the training labels for this task indicate whether a collection of sets contains a hitting set or not. The end-to-end loss is therefore the binary cross-entropy loss. We train with 1502 data points, and report the test accuracy over 316 data points. In the recursive arithmetic tasks, the output layer of the reasoning network is the last layer in the MLP, and the MAE is used as a regression loss function. We train and test on 10000 data points, of varying list lengths 2–5. For lists shorter than 5, we pad the list with -1 values, to ensure all the lists are of the same length. This is required as

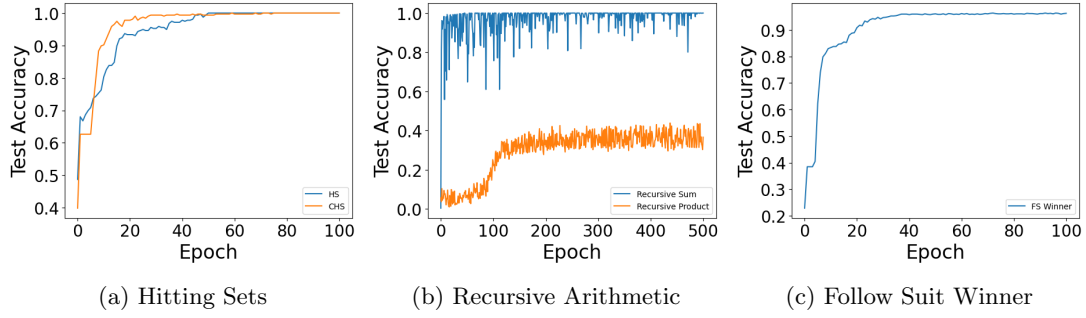


Figure 5.9: Test set accuracy results when training neural surrogate models to learn complex knowledge.

the architecture of the neural network expects a fixed size as input. Finally, in the Follow Suit Winner task, we use a softmax output layer with the Cross-Entropy Loss, and train and test on 10,400 and 1000 data points, respectively. The results are shown in Figure 5.9, and demonstrate we are able to train the neural network to a near perfect accuracy, using randomly generated input data evaluated using the given hypothesis, in all tasks except the Recursive Product task in Figure 5.9b. This is a hard task where the range of output values varies significantly more than the Recursive Addition task. We assume a prediction is correct only if it matches the ground-truth label exactly. It is likely a larger network with more data would be needed on this task. Note that although the trained networks largely perform well, they have not necessarily learned the general knowledge expressed by the symbolic hypothesis. However, in this case, as long as the network can correctly emulate the function of the hypothesis for the training data, this is sufficient to train our mapping function in an end-to-end fashion.

Let us now present the results of training the mapping function, given a pre-trained reasoning network with frozen weights. Recall that we also assume a perception component with perfect clustering, to evaluate whether our mapping function can be trained in the best case, where no noise is present. We use three tasks in our evaluation: (1) MNIST Addition, with different datasets containing 2–5 single digits in each example. (2) Playing Card Addition, which is the same reasoning task as MNIST Addition, except we represent digits with playing cards, and therefore they can take values 0–51, instead of 0–9. We evaluate with different datasets ranging from 2–4 playing cards in each example, which introduces significantly more choice than the MNIST Addition task. (3) Follow Suit Winner, with 4 playing cards in each example. This task has even more choice, because there are many combinations of different playing cards that satisfy each downstream label. For each task, we randomly generate a synthetic concept to cluster assignment mapping, and transform datasets containing ground-truth concept values to yield datasets that simulate cluster assignment predictions. We then evaluate the end-to-end task accuracy, and also inspect the permutation matrix output from the mapping network, when given the synthetic concept to cluster assignment mapping as input. In this case, if the mapping has been learned correctly, we expect a perfect permutation matrix in the form of an identity matrix as output.

For example, in the MNIST Addition task, we generate cluster datasets by applying the following randomly generated mapping:

- Concept 0 $\rightarrow$ Cluster 4
- Concept 1 $\rightarrow$ Cluster 3
- Concept 2 $\rightarrow$ Cluster 2

- Concept 3 $\rightarrow$ Cluster 6
- Concept 4 $\rightarrow$ Cluster 7
- Concept 5 $\rightarrow$ Cluster 8
- Concept 6 $\rightarrow$ Cluster 9
- Concept 7 $\rightarrow$ Cluster 0
- Concept 8 $\rightarrow$ Cluster 1
- Concept 9 $\rightarrow$ Cluster 5

The mapping function should then learn a permutation matrix that effectively inverts this mapping, to output concept values given cluster assignments. Once trained, consider the following batch of vectors as input to the mapping function, corresponding to our randomly generated concept mapping:

$$\begin{aligned} &[[0, 0, 0, 0, 1, 0, 0, 0, 0, 0], \\ &[0, 0, 0, 1, 0, 0, 0, 0, 0, 0], \\ &[0, 0, 1, 0, 0, 0, 0, 0, 0, 0], \\ &[0, 0, 0, 0, 0, 0, 1, 0, 0, 0], \\ &[0, 0, 0, 0, 0, 0, 0, 1, 0, 0], \\ &[0, 0, 0, 0, 0, 0, 0, 0, 1, 0], \\ &[0, 0, 0, 0, 0, 0, 0, 0, 0, 1], \\ &[1, 0, 0, 0, 0, 0, 0, 0, 0, 0], \\ &[0, 1, 0, 0, 0, 0, 0, 0, 0, 0], \\ &[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]] \end{aligned}$$

We expect to see a perfect permutation matrix in the form of the identity matrix as output. To simplify notation, we will collapse the one-hot vectors into a single integer denoting the index in the one-hot vector which has the value 1. In this case, our input to the mapping function during evaluation is $[4, 3, 2, 6, 7, 8, 9, 0, 1, 5]$, and we expect the mapping function to output $[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]$ during evaluation. In all experiments, unless otherwise stated, we set the value of the hyper-parameter $\lambda = 20$ as this returned the best results, out of a set of possible values $\lambda \in \{1, 2, 5, 10, 20, 50\}$.

MNIST Addition. We pre-train the reasoning network with 10,000 randomly generated data points, and train the mapping network with 20,000 examples end-to-end (i.e., a sequence of raw images together with a downstream label). We train both networks for 10 epochs with a batch size of 8. In this task, we learn the perfect mapping from clusters to concepts for all datasets. The end-to-end task accuracy is 0.99995, 0.99515, 0.97835, and 0.9713 for datasets with 2, 3, 4, and 5 digits respectively. Note the decrease in accuracy is due to the decreasing performance of the reasoning network, rather than the cluster to concept mapping. The configuration was chosen as what was required to achieve good performance on this task.

Playing Card Addition. In this task, we train both networks with 50,000 examples, for 2000 epochs with a batch size of 128. This is sufficient to learn the perfect mapping for all datasets, outputting $[0, 1, 2, \dots, 49, 50, 51]$ given the concept-to-cluster mapping used to generate

the datasets as input. The end-to-end task accuracy is 0.9978, 0.73906, and 0.7161 for datasets with 2, 3, and 4 playing cards respectively. Again, the errors are due to mistakes in the reasoning network, rather than an incorrect mapping. Whilst it may be possible to improve the accuracy of the reasoning network by generating more synthetic data, and training for longer, we have achieved the goal of obtaining the correct mapping. Recall that ultimately, the correct symbolic hypothesis is used to perform inference, and the reasoning network is just a differentiable tool used to learn the mapping function. Assuming a perfect clustering, perfect accuracy is achieved for all datasets, when the clustering network, the learned mapping, and the symbolic hypothesis are used. As a benchmark, we can compare these results to Embed2Sym’s cluster optimisation stage. As discussed in Section 4.4.2 in Chapter 4, Embed2Sym failed to solve the task with 4 playing cards, as the ASP solver Clingo became stuck in the grounding stage.

Follow Suit Winner. In this task, we pre-train the reasoning network with 100,000 randomly generated examples, and train the mapping network with 50,000 examples end-to-end. We select this configuration as a reasonable upper-bound on the size of the datasets. We train both networks for 500 epochs with a batch size of 64. In this task, we fail to learn the correct mapping, and do not output a perfect identity matrix during evaluation. This is best analysed by plotting a 2-D heatmap of the output matrix, where the shaded regions indicate entries of the value 0, and the yellow colour indicates an entry of the value 1. The results are shown in Figure 5.10. Figure 5.10a shows the expected result of a perfect identity matrix. You can see in Figure 5.10b, whilst there is a diagonal trend in the matrix, the diagonal is not perfect, and multiple cluster assignments are mapped to the same concept value. This is because the constraints imposed by the reasoning network in this task are not strong enough to learn the correct mapping. For example, assuming a single deck of cards, there are 4,523,376 possible combinations of playing card values across the four players where player 1 is the winner. For the other players, there are 658,008 combinations. Therefore, it is very challenging to learn the correct mapping given a dataset of only 50,000 examples. It is possible that a dataset enumerating all of these combinations would have to be observed, in order to separate the cards perfectly. Also, in this task, the mapping network can learn to map clusters of similarly ranked card values to the same concept value, and still achieve a reasonable level of accuracy. For example, when $\lambda = 20$, the mapping network outputs 12 distinct concept values, and we achieve an end-to-end accuracy of 0.66478 when the ground-truth hypothesis is used instead of the reasoning network. Interestingly, we observe that when we set $\lambda = 0$, which effectively removes the permutation loss, and the network optimises only the end-to-end objective, the accuracy increases to 0.77312 with 5 distinct concept values (see Figure 5.10c). Whilst this shows our permutation loss has the desired impact of encouraging the mapping network to map each cluster assignment to a greater number of unique concept values, the results clearly indicate a failure case, when trying to learn a concept mapping in tasks with a large number of choices for the possible values of the primitive concepts. We hypothesise that this is a fundamental limitation of training end-to-end with a concept bottleneck, as the network is highly constrained. There simply is not a strong enough back-propagation signal in this task to enable the mapping network to learn a perfect clustering. Without a high-dimensional embedding space, we hypothesise the network struggles to learn representations for complex knowledge.

To investigate this, we conduct an ablation study and demonstrate training an end-to-end architecture on the Follow Suit Winner task, with progressively fewer architectural restrictions applied to each component during training. If our hypothesis is correct, we expect the end-to-end accuracy to improve with fewer restrictions. For this analysis, instead of using a clustering network, we fine-tune a ResNet on the “Rank Suit Crop” images introduced in Section 5.1. We

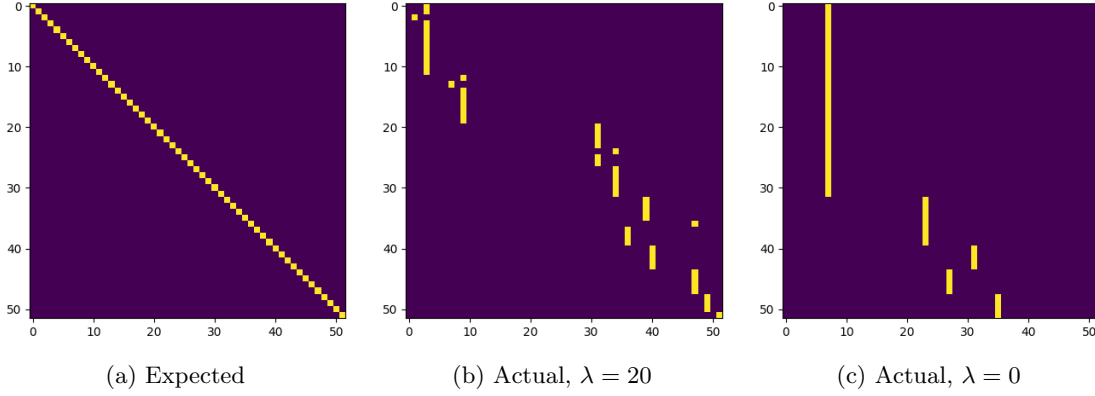


Figure 5.10: Expected vs. Actual matrices output from the trained mapping network on the Follow Suit Winner task.

use pre-trained weights for the ResNet, obtained using the ImageNet dataset, and then freeze the weights following fine-tuning. This network achieves 100% accuracy when classifying playing card images, which enables us to remove the perception network as a variable, when considering the ultimate task performance. We evaluate three architectural variants with progressively fewer restrictions:

1. **(Fully Constrained).** We first connect the frozen ResNet to a mapping MLP with a Gumbel Softmax layer as output. This is connected directly to a pre-trained reasoning network with frozen weights, to perform a classification of the downstream label. This represents a highly constrained architecture which resembles the cluster to concept mapping architecture in Figure 5.8. The reasoning network is a 3-layer MLP, and performs perfectly with 100% accuracy. It is trained using randomly generated one-hot vectors denoting concept values, and the correct downstream label. The only trainable component is the mapping MLP, which has to map feature vectors output from the ResNet, into one-hot vectors denoting the value of the concepts.
2. **(Without Gumbel-Softmax).** The second experiment replaces the Gumbel-Softmax layer in the architecture from experiment 1 with a normal softmax layer. This removes the restriction that the mapping MLP has to output perfect one-hot vectors. Note the reasoning network remains frozen. This corresponds to the architecture presented in Figure 5.7.
3. **(Without a bottleneck, without freezing the reasoning network).** The final experiment removes the concept bottleneck entirely, and connects the output of the ResNet directly to a trainable reasoning network, where the embedding size as input to the reasoning network is 2048×4 (given 4 players), instead of 52×4 with the concept bottleneck applied. This is an under-constrained architecture, and the network is able to learn a complex representation in the latent embedding space to map inputs directly to outputs, avoiding the requirement to predict intermediary concepts via a concept bottleneck.

We then measure the end-to-end task accuracy on a held-out test set of 10,000 examples. The results are shown in Table 5.1, and indicate that with fewer restrictions during training, the network is able to perform more accurately, i.e., interpretability is sacrificed to obtain a reasonable level of performance. This highlights the challenge of trying to train an architecture with a concept bottleneck in an end-to-end fashion, even when everything is differentiable.

Architecture	End-to-End Accuracy
Fully Constrained	0.1011
Without Gumbel-Softmax	0.6974
Without a bottleneck, without freezing the reasoning network	0.9013

Table 5.1: End-to-end task accuracy on the Follow-Suit Winner task with decreasing architectural restrictions during training. With fewer architectural restrictions, accuracy increases.

To summarise, we have introduced a novel architecture for labelling the clusters, which reduces the end-to-end training requirement to be as simple as possible, where only an MLP needs to be trained to map clusters to concepts. Whilst we have outperformed Embed2Sym on the Playing Card Addition task, we failed to learn a complete cluster labelling on the Follow Suit Winner task, where there is a large amount of choice for the possible values of the concepts. Also, we have demonstrated that training end-to-end when concept restrictions are applied is challenging, and that removing the concept restrictions results in a higher level of performance. Therefore, we conclude differentiable techniques for labelling the clusters are unlikely to scale to more complex tasks with a large amount of choice for the latent concept values.

Our aim so far has been to develop a more scalable plug-in replacement for NeurASP, such that we could scale our NSIL approach introduced in Chapter 4 to more complex problems. We have shown that scaling neuro-symbolic reasoning is very challenging, even in the best case when everything is differentiable. Also, even if we did replace NeurASP with a more scalable alternative, there are still some challenges with the NSIL algorithm. We have shown in Chapter 4 that NSIL is not guaranteed to learn an optimal hypothesis, and also, NSIL relies upon computing an expensive abduction of possible concept values, when creating a set of corrective examples. We therefore conclude that pursuing a plug-in replacement for NeurASP within the NSIL architecture is not worth it, and a different approach is needed. Note that we are still interested in pursuing an approach where the training of the perception component is decoupled from the reasoning task, in order to utilise the advancements in unsupervised, and self-supervised learning. As a result, we now attempt to label the clusters, whilst also learning the symbolic hypothesis, in a manner in which we can avoid computing the abduction of all the concept values, whilst also providing an optimality guarantee.

5.2.2 Breadth-First Search

In contrast to the previously explored differentiable cluster labelling approaches, we instead model the cluster labelling problem as a purely symbolic breadth-first search over the possible concept values, progressively constructing a mapping from clusters to concepts. We choose to explore a method of solving this problem symbolically, so we can learn the hypothesis of the reasoning task simultaneously, and also formally guarantee that the cluster to concept mapping, as well as the learned hypothesis, are correct with respect to the training data. The idea is to assume a partial mapping, and learn a candidate hypothesis with a LAS symbolic learner that allows choice over un-assumed concept values in the mapping. If no hypothesis can be learned with a partial set of assumptions (and full choice over the remaining cluster to concept assignments), then it must be the case that the assumed partial mapping is incorrect, since the LAS symbolic learners are guaranteed to learn an optimal hypothesis. This enables us to make a small number of assumptions, and prune vast areas of the search space if no hypothesis can be learned.

The search can be visualised as a tree, which is shown in Figure 5.11 for the MNIST Addition

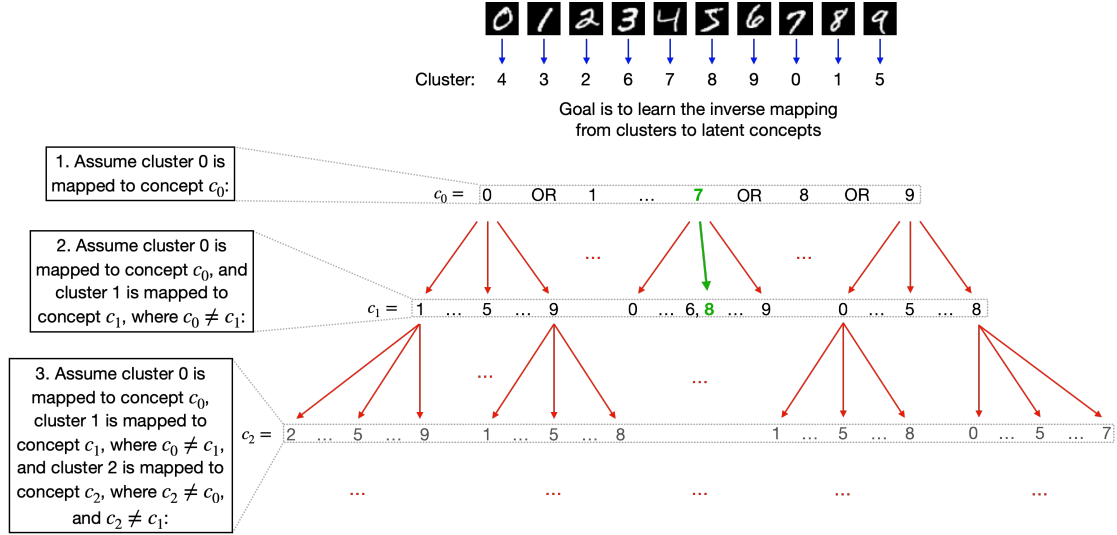


Figure 5.11: Breadth-first search to label the clusters, shown with the MNIST Addition task. The green path shows the correct mapping.

task. The execution of our method moves left to right in a breadth-first manner, starting with the top-left node in the tree. The nodes at level i in the tree assume cluster i maps to the concept value c_i . For example, at level 0 in the MNIST Addition task, there are 10 nodes. Each node assumes cluster 0 is mapped to a different concept value (i.e., $c_0 = 0, c_0 = 1, \dots$, or $c_0 = 9$). At level 1, there are 9 child nodes under each parent assumption at level 0, representing the possible values of the next cluster assignment. Note there are 9 nodes because there are 9 remaining clusters, given a parent assumption of the assignment of cluster 0 (in the MNIST Addition task). If the tree in Figure 5.11 were to be expanded in full, it would represent all possible cluster to concept mappings. Clearly, enumerating this tree in full is infeasible, so we adopt the symbolic learner to help us prune the tree as we construct it from the root to the leaves.

Symbolic Learning Task At each node, we create a symbolic learning task, using the training data and the assumed cluster to concept mapping in the lineage of that node (i.e., all the assumed cluster assignments in the path from the root to the node). Since the LAS systems support choice rules in the example context, this enables us to leave the remaining cluster assignments as a choice for the symbolic learner. The assumed cluster to concept assignments are represented as facts, by simply transforming each cluster prediction in the training data into a concept using the assumed mapping. To create the learning task, we create a set of training examples, where the context contains facts regarding the assumed cluster assignments, and choice rules for any un-assumed cluster assignments. The training examples have infinite weight penalties, which guarantees the symbolic learner will learn a hypothesis that covers all of the examples, or output an error indicating there is no hypothesis in the search space that covers all of the examples. The final task label in the training data is encoded into the inclusion and exclusion sets, as usual. As more clusters are assigned a concept value, there are more examples for the symbolic learner without choice rules, so the learning time of each task reduces.

Note that if the assumed cluster to concept mapping is correct at this node, then it is guaranteed that the symbolic learner will be able to learn a hypothesis that covers all of the training examples, since the correct remaining concept values will be present in the choices for clusters yet to be assigned a concept value. Conversely, if the symbolic learner fails to learn a

hypothesis, then the assumed cluster to concept mapping must be incorrect, because there are no remaining choices for the concept values of the un-assumed clusters that lead to a hypothesis being learned. It is this property that enables us to prune the search space of the cluster to concept mapping, whilst guaranteeing the learned mapping, and the hypothesis are correct with respect to the training data. If the symbolic learner fails to learn a hypothesis at a given node, we know this assumption, and all future child assumptions are incorrect, so we can mark this node as one which won't be expanded at the next level. In Figure 5.11, the expanded path is shown in green. Only nodes which result in a hypothesis being learned are expanded at the next level in the tree.

At this stage, one may wonder whether one could A) Inspect the choice made by the symbolic learner for the values of the remaining cluster assignments to act as a speed-up, and B) Whether the learned rules could be utilised to perform an abduction at later levels in the tree, further restricting the choice of the remaining clusters. Unfortunately, neither of these extensions would preserve the optimality guarantee. Option A is not possible, since the LAS systems compute coverage on a per example basis, i.e., $H \cup B \cup e_{\text{ctx}} \models e_{\text{inc}}$, and $H \cup B \cup e_{\text{ctx}} \not\models e_{\text{exc}}$, and therefore make a choice over remaining concepts locally within each example. We did attempt adding constraints to the background knowledge to force the symbolic learner to make consistent choices for the remaining cluster values, but this significantly impacts the performance, as the full combinatorial search of the remaining assignments is effectively delegated to the symbolic learner. Option B is also not possible, because there is no guarantee the hypothesis learned at a given node will be correct, since the mapping is incomplete. There could be multiple hypotheses in the search space that cover all of the examples with a partial mapping, and only upon consideration of a complete mapping can you determine whether the hypothesis is correct with respect to the training data.

MNIST Addition Example Consider the first node in the tree in Figure 5.11 for the MNIST Addition task, which assumes $c_0 = 0$, i.e., cluster 0 is mapped to concept 0. We then find all data points in our training dataset, which contain a cluster prediction 0. We then substitute the value of cluster 0 with the concept assumption, whilst creating a choice rule for the concept values of any un-assigned clusters. A training example for the LAS symbolic learner takes the following form:

```
#pos(id1, { result(10) }, { }, {
  digit(1,0).
  1 { digit(2,1); digit(2,2); ... digit(2,9) } 1.
}).
```

In this case, the inclusion set contains the downstream label, which is 10 for this data point. We include a constraint in the background knowledge to ensure only one label can be covered for a given pair of digit values, instead of specifying all other labels in the exclusion set. The context contains the assumption that the first digit in the data point has the concept value 0 (obtained from our assumed mapping), and the second digit value is represented as a choice rule, since this has a cluster prediction yet to be assigned a concept value. Note that we can remove the choice of concept 0 from the possible values for the second digit.

Experimental Setup We evaluate our proposed method on the MNIST Addition and E9P tasks from Chapter 4, by generating a training dataset of cluster predictions for each MNIST digit image. Since the purpose is to evaluate our method in the best-case scenario, this dataset assumes perfect clustering with no noise, so each digit value receives a unique cluster prediction.

The downstream label in both the Addition and E9P tasks remains the same as in Chapter 4. For this experiment, each symbolic learning task contains 20 examples, that are selected from a dataset of 20,000 randomly generated data points. The algorithm for selecting the data points to create the examples is as follows. We iterate over the full dataset, and select a data point if it contains a cluster assignment that has an assumed entry in the partial mapping at the current node, until we reach 20 selected data points. This ensures that we do not give any examples to the symbolic learner with a full choice over the concept values for all concepts in the data point (e.g., for both digits in the MNIST Addition task). Note this is simply a performance improvement for our experiments, and the general method does not require such specific dataset selection. Given our training set, we run our breadth-first search approach and inspect the learned hypothesis and the complete cluster to concept mapping at the final explored node in the tree.

Results and Analysis We are able to learn the correct hypotheses, and the correct cluster to concept mapping on the MNIST Addition and E9P tasks. The total learning time is approximately 2 minutes, and the search tree is effectively pruned to reduce the computation of the full cluster to concept mapping search space. Whilst this method succeeds on the 2-digit MNIST Addition and E9P tasks, for tasks with a large number of concept values, such as Follow Suit Winner, this method takes a significantly longer time to complete. Analysing our approach further, it is apparent that in the best case, where only one possible concept assignment assumption remains at each level in the tree, the complexity is $k + k - 1 + k - 2, \dots$, in terms of the number of visited nodes, and therefore the number of symbolic learning tasks required. This is equivalent to $\frac{k^2+k}{2}$, where k is the number of concept values.

In the MNIST Arithmetic tasks, this results in $\frac{10^2+10}{2} = 55$ nodes to explore in the best case, whereas in the Follow Suit Winner task, the best case number of nodes is $\frac{52^2+52}{2} = 1378$, since there are 52 possible values for each playing card. Whilst this is not poor, in practice, it is very unlikely for the best case to occur, as there are multiple concept assumptions remaining at each level in the tree, especially early in the search, when the symbolic learner is able to cover the examples with a greater choice of concept values. This is especially the case in the Follow Suit Winner task, as there are many possible choices for the values of the concepts that correctly output each downstream label. As a result, the number of visited nodes is significantly greater than 1378.

Also, it is not clear how this method would perform in the case of noisy clustering, which is a more realistic assumption in a practical setting. The examples for the symbolic learner would have to be weighted, which would likely have a significant impact on the run-time at each node in our search tree, as there would be many more hypotheses that the symbolic learner would have to consider. It is also likely that the decision to prune descendant nodes would have to be modified with weighted examples, since the symbolic learner may learn a hypothesis that covers *at least one* example under each possible concept assignment at a certain level. In the worst case this would result in our method performing a brute-force search. To mitigate this, the decision to prune descendant nodes if there is no hypothesis learned could be modified to prune descendant nodes if the learned hypothesis does not cover an acceptable number of examples. However, doing so sacrifices the optimality guarantee, since this approach may inadvertently prune the correct solution. As a result, although we have developed a more robust method that is guaranteed to learn an optimal hypothesis, given perfect clustering, this method is unfortunately not scalable beyond problems with a small number of primitive concept values.

5.3 Summary

In this chapter, we have explored a variety of new ideas for scaling a neuro-symbolic integration via decoupling the perception component from the reasoning task. We first demonstrated the potential of unsupervised, and self-supervised techniques, for clustering the raw input data without concept labels. However, there does not appear to be a single method that performs well on a wide range of datasets. Also, adopting a clustering network for the perception component poses a challenge for tasks that require more advanced perception such as object detection, as the system is limited to extracting concepts via image classification.

Also, we developed two novel methods for labelling the clusters with concept values. In doing so, we assumed perfect clustering from the perception component. Our first method was a fully differentiable architecture, and we demonstrated some incremental improvements compared to the state-of-the-art approach Embed2Sym. However, we ultimately showed this method, and end-to-end differentiable methods with a concept bottleneck in general, struggle to train successfully given complex tasks with a larger number of choices for the possible values of the concepts. The second method labels the clusters and learns a hypothesis via a symbolic breadth-first search. In this noise-free setting, i.e., assuming perfect clustering, this method is guaranteed to learn an optimal hypothesis, provided one exists in the hypothesis space. Also, the method does not rely upon a neuro-symbolic reasoning approach to train the perception component. However, we have also demonstrated this method does not scale well with respect to the number of possible concept values.

In summary, training a perception network using a signal from downstream labels is challenging, especially when there is a large number of possible concept values. Unsupervised and self-supervised approaches that rely on clustering are limited to tasks that can rely on image classification, and suffer from the assignment problem to label the clusters. This is exacerbated when the hypothesis to perform the reasoning task has to be learned. Alternatively, manually labelling a large dataset for training a perception network in a fully supervised fashion is also not feasible. Therefore, we consider whether additional world knowledge regarding unstructured inputs can be leveraged to train the perception component to a high accuracy in a data-efficient manner? In particular, large vision-language models could be relevant in this endeavour. This is explored in the next chapter.

Chapter 6

Scalable Neuro-Symbolic Learning and Reasoning with Foundation Models

Given the scalability challenges encountered in the last chapter, it is worth taking a step back, and asking ourselves whether we should even pursue the goal of learning in an end-to-end fashion. Whilst a complete end-to-end approach is highly flexible, the user ultimately sacrifices control over the learning process, since they only provide input and output pairs. A complete architecture is therefore free to learn any representation that optimises the training objective. This poses a significant challenge for AI safety, as an undesirable representation may be learned, whilst perfectly fitting the training data. It is impossible to guarantee such a system will behave as expected, when faced with situations different to those in which it has been trained.

We have shown that a complete fully neural end-to-end architecture requires a larger embedding space to ensure high performance (see the Follow Suit Winner experiments in Section 5.2). This is undesirable, because the learned representations are difficult to interpret. Adding in a differentiable concept bottleneck layer (Koh et al., 2020) between the perception and reasoning components enables the user to exercise some control, as the perception network is forced to predict a set of primitive concepts, which makes it easier to interpret the behaviour of the reasoning network during inference. However, we have shown these architectures may not perform as well as those without the concept bottleneck, and there is no guarantee that desirable concept representations will be learned (Margeloiu et al., 2021).

End-to-end *neuro-symbolic* learning approaches increase the level of control, as they learn interpretable hypotheses that perform the reasoning task, defined in terms of primitive concepts that are predicted by the neural network. The hypotheses can then be modified after training to either correct mistakes, or solve a different reasoning task, provided the set of primitive concepts remain unchanged. In these situations, the neural network does not need to be retrained. The drawback is that these approaches are difficult to scale, and, as with fully neural concept bottleneck models, there is no guarantee the perception component will learn desirable concept representations. This is also true of neuro-symbolic reasoning systems, which enable even greater control through a manual specification of the knowledge required to solve the reasoning task. (Marconato et al., 2024) have demonstrated that such systems can exploit *reasoning shortcuts*, which can have severe consequences during inference. For example, in autonomous driving (see Figure 6.1), if the reasoning task contains the knowledge “*STOP if there is a pedes-*

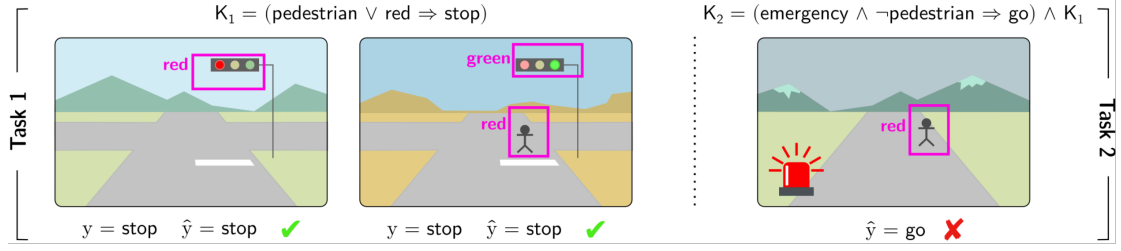


Figure 6.1: Example reasoning shortcut in the autonomous driving domain. Source: (Marconato et al., 2024).

trian or a red light”, and the perception network is trained to classify images as containing a *pedestrian*, a *red light*, or a *green light*, there is nothing to prevent the neural network learning to classify pedestrians as red lights, even with an exhaustive training set. This is due to a reasoning shortcut: The system can correctly predict “STOP”, if the neural network classifies an image as containing a red light, in cases where the light is green and there is a pedestrian present. Consider now during inference of a downstream task, where the knowledge base contains the additional rule: “If there is an emergency and there is NOT a pedestrian, then GO”. In the case where there are no traffic lights, and a pedestrian is present, the system would incorrectly predict “GO”, if the neural network mis-classifies the pedestrian as a red traffic light. Due to these reasoning shortcuts, which can be difficult to identify a priori, even neuro-symbolic systems can be difficult to control, when learning in an end-to-end fashion. With this in mind, we propose a set of desiderata for a neuro-symbolic system. An ideal system should:

1. Scale and perform strongly for a wide-range of complex tasks, learning optimal solutions.
2. Learn interpretable solutions.
3. Enable fine-grained control in high-stakes applications.
4. Require a minimal amount of user effort, in terms of the amount of labelled data required, and the task-specific engineering.

Considering our two main systems introduced so far, FF-NSL (Chapter 3) and NSIL (Chapter 4), FF-NSL can scale more easily, is guaranteed to learn an optimal set of rules with respect to the neural network predictions, and it is easier to control, since the perception and reasoning components are trained sequentially. However, the main drawback is that FF-NSL requires a large labelled dataset for training the perception component. NSIL on the other hand, eliminates the requirement for labelling the primitive concept values, but it is difficult to scale to complex tasks, and it is not guaranteed to learn an optimal solution. It is also more difficult to control, since it is trained end-to-end, and thus vulnerable to learning reasoning shortcuts. Both systems learn interpretable symbolic representations, but require a lot of task-specific engineering, including bespoke perception components (i.e., the architecture of the CNN needs to be defined, which is often task specific). Note that the desire to reduce the engineering effort, and the ability to exercise control, are somewhat conflicting properties. We believe the ideal system should support both extremes, and the user should be able to trade these off for the desired application domain.

The research challenge that we address in this chapter is how to develop a system that meets all of the desiderata outlined above. At this point, there are two paths forward. We could either: (1) Continue in our efforts to scale NSIL, but this is fraught with challenges, as

we have demonstrated in Chapter 5. Or, (2), we could attempt to improve FF-NSL, since it is inherently more scalable, it is guaranteed to learn an optimal hypothesis given sufficiently accurate neural network predictions, and it enables finer-grained control over what is learned, since the perception and reasoning components are trained sequentially. Sequential training is a natural decoupling of the two components, and breaks the combinatorial explosion encountered in end-to-end systems, when dealing with the choice of possible values for the concepts. The outstanding question to address is, how can we mitigate the requirement for extensive data labelling, and manual engineering?

To mitigate the data labelling and engineering requirements, a natural question to ask is whether we can utilise recent advancements in large foundation models (Bommasani et al., 2021), especially those trained with joint vision and language representations, e.g., (Li et al., 2022b; Radford et al., 2021), since these models have large amounts of implicit knowledge embedded within their latent space. However, it is not immediately obvious how to integrate these models with symbolic learning and reasoning, since these architectures are often trained via tasks with a natural language component such as VQA, image captioning, or image-text matching.

In this chapter, we develop a novel method, called NeSyGPT, which integrates a vision-language foundation model with a symbolic learner. The novelty includes the method of generating questions and answers for fine-tuning the vision-language foundation model, given primitive concept labels, and also transforming the answers during inference, into a symbolic representation for learning a hypothesis. We demonstrate these components can be implemented manually, to exercise greater control, and we also develop the necessary prompts to enable a LLM to generate them automatically, which reduces the engineering effort in cases where autonomy is more desirable. NeSyGPT fine-tunes the BLIP vision-language foundation model (Li et al., 2022b) to extract concepts from raw data, before learning a hypothesis to solve a downstream task. The architecture utilises the implicit knowledge of BLIP, and therefore, NeSyGPT only requires a few labelled data points to fine-tune for a specific task. We evaluate NeSyGPT in four problem domains, each with specific challenges, and demonstrate that it out-performs a variety of neural and neuro-symbolic baselines. Specifically, compared to all of the studied baselines, NeSyGPT offers the following advantages:

1. It requires a reduced amount of labelled data points to extract primitive concepts from raw data, and learn the hypothesis of the reasoning task.
2. It scales to complex tasks that have a large number of primitive concept values.
3. It supports both the learning of and the reasoning over the symbolic hypothesis.
4. It can solve tasks that require detecting multiple objects and their properties within a single image, e.g., the CLEVR-Hans3 dataset (Stammer et al., 2021), thus offering more advanced perception ability compared to other approaches that rely upon a CNN to perform image classification.
5. It can utilise a LLM to generate questions and answers for fine-tuning BLIP, and to construct the programmatic interface between the neural and symbolic components. This reduces the engineering effort compared to a manual specification.

Since the perception and reasoning components are decoupled and trained sequentially, NeSyGPT offers more control than end-to-end systems, and avoids the reasoning shortcut problem identified by (Marconato et al., 2024). NeSyGPT is also guaranteed to learn an optimal

hypothesis with a given set of predictions from BLIP, since the LAS symbolic learners are guaranteed to learn an optimal hypothesis in the presence of noisy examples (Law, 2018) (see also Chapter 3, which demonstrates the LAS systems have strong performance when learning from neural network predictions). The remainder of this chapter is organised as follows. In Section 6.1, we formally define the problem specification. In Section 6.2, we present the NeSyGPT framework, and give example instantiations for each of the considered learning tasks. In Section 6.3, we present our comprehensive evaluation, and demonstrate NeSyGPT achieves state-of-the-art performance when learning with a minimal set of labelled data points. We also evaluate the capability of a LLM for automating the manually engineered components of the NeSyGPT architecture, and investigate the performance of more recent vision-language foundation models when used in a zero-shot fashion, since these models do not currently support fine-tuning via concept labels. The work in this chapter was published at the 18th International Conference on Neural-Symbolic Learning and Reasoning (Cunnington et al., 2024), where it was selected for a spotlight presentation.

6.1 Problem Formulation

We again decompose the overall problem into perception and reasoning functions, where the knowledge for the reasoning function is learned symbolically. However, in this chapter, the perception function performs VQA, and is defined by $f : \langle \mathcal{X}, \mathcal{Q} \rangle \rightarrow \mathcal{A}$, where $\mathcal{X}$, $\mathcal{Q}$, and $\mathcal{A}$ are the spaces of raw image inputs, natural language questions, and answers, respectively. The answers represent the concept values present within an image. The reasoning function $h : (\mathcal{A}^u \times \mathcal{M})^n \rightarrow \mathcal{Y}$ maps n pairs of answers and meta-data, to a target label. Note that there could be u questions asked of each image, and therefore, u answers for each pair. The meta-data in the space $\mathcal{M}$, is used to encode any extra information associated with the raw input. $\mathcal{Y}$ is the space of the target labels.

In this chapter, we learn f and h sequentially, where h is learned using predictions from f , once f is learned. Therefore, we use two datasets: (1) $\mathcal{D}_{\text{image}} = \{\langle \mathbf{x}, z \rangle, \dots\}$, where $\mathbf{x} \in \mathcal{X}$ is an image, and $z \in \mathcal{Z}$ is a concept label, and (2) $\mathcal{D}_{\text{task}} = \{\langle X, M, y \rangle, \dots\}$ where $X = \langle \mathbf{x}_1, \dots, \mathbf{x}_n \rangle$ is an n -length sequence of images, $M = \langle m_1, \dots, m_n \rangle$ is an n -length sequence of meta-data associated with X , and $y \in \mathcal{Y}$ is a target label. Since f is a perception function that performs VQA, we require, in addition to the raw input, sets of questions and answers. The questions and answers can be manually engineered, or generated automatically using the concept label z , and are assumed constant for a given task. We denote the set of questions as $Q \subseteq \mathcal{Q}$ and the set of possible answers for a given question $q \in Q$ as $\mathbf{A}_q \subseteq \mathcal{A}$. For example, in the Follow Suit task (see Figure 6.2), the question $q = \text{"which rank is this?"}$ has the set of possible answers $\mathbf{A}_q = \{\text{"2"}, \text{"3"}, \dots, \text{"Ace"}\}$. To link the questions and answers to the concept label, we require a look-up table $l : \langle \mathcal{Q}, \mathcal{Z} \rangle \rightarrow \mathcal{A}$ which is used to map question and concept label pairs to the correct answer. For example, given a concept label $z = \text{"Jack of Hearts"}$, the look-up table returns the answer "Jack" for the question $q = \text{"which rank is this?"}$, and the answer "Hearts" for $q = \text{"which suit is this?"}$. The look-up table can be constructed manually or automatically, within the question and answer generation. Note this is only used to obtain the necessary data for learning the perception VQA function.

The first goal is to learn f such that $\forall \langle \mathbf{x}, z \rangle \in \mathcal{D}_{\text{image}}$, and $\forall q \in Q$, $f(\mathbf{x}, q) = l(q, z)$, i.e., f returns the correct answer for all images and all questions. Once f is learned, the next objective is to learn h . Let us define the sequence of predicted answers for an image $\mathbf{x}_i \in X$

as $\hat{A}_i = \langle f(\mathbf{x}_i, q_1), \dots, f(\mathbf{x}_i, q_u) \rangle$, where $q_j \in Q$. The second goal is to learn h , such that $\forall \langle X, M, y \rangle \in \mathcal{D}_{\text{task}}, h(\langle (\hat{A}_1, m_1), \dots, (\hat{A}_n, m_n) \rangle) = y$. Intuitively, this means that each data point in $\mathcal{D}_{\text{task}}$ has a correct downstream prediction. For example, in Figure 6.2, assuming correct predictions, the answers are $\hat{A}_1 = \langle \text{"2"}, \text{"Hearts"} \rangle$, $\hat{A}_2 = \langle \text{"9"}, \text{"Diamonds"} \rangle$, $\hat{A}_3 = \langle \text{"Jack"}, \text{"Hearts"} \rangle$, and $\hat{A}_4 = \langle \text{"Ace"}, \text{"Clubs"} \rangle$, and the meta-data M indicates which player each card corresponds to, e.g., $m_1 = \text{"Player 1"}$. The target label is $y = 3$, as Player 3 is the winner. To learn f and h we propose the NeSyGPT architecture, which consists of the BLIP Fine-Tuning phase for learning f , and the Symbolic Rule Learning phase for learning h . The overall architecture is shown in Figure 6.2.

6.2 NeSyGPT Framework

6.2.1 BLIP Fine-Tuning

Given an image data point $\langle \mathbf{x}_i, z_i \rangle \in \mathcal{D}_{\text{image}}$, and a question $q \in Q$, the answer returned from the look-up table $l(q, z_i)$ is used to fine-tune BLIP, alongside the image x_i . The question-answer generator is specific to each task, and can be manually engineered, or obtained automatically using a LLM. To fine-tune BLIP with the generated questions and answers, we use the “VQA” mode, where the pre-trained BLIP model is rearranged by encoding an image-question pair into multi-modal embeddings, before passing the embeddings to a text decoder to obtain the answer. BLIP is then fine-tuned end-to-end using a cross-entropy loss with ground-truth answers. Let us now present our instantiations of the question answer generator, for each task considered in our evaluation.

Two-Digit Arithmetic. We first consider the two-digit MNIST Arithmetic tasks from Chapter 4, which includes the Addition and E9P variants. The question-answer generator takes as input each image label $z \in \{0 \dots 9\}$, and creates an entry in the lookup table:

$$l[\text{"which digit is this?"}, z] = z$$

In this task, there is only one question $Q = \{\text{"which digit is this?"}\}$, and $\mathbf{A}_{q_1} = \{0 \dots 9\}$. The lookup-table is then used to retrieve the answer for fine-tuning BLIP, given the question “*which digit is this?*”, and the image label z . Note in this task the given label z is the same as the answer to the question, due to the simplicity of the task.

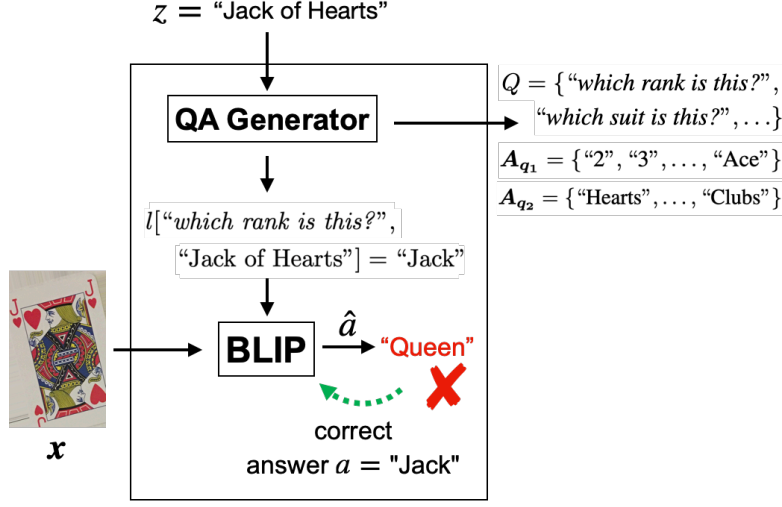
Follow Suit Winner. In the Follow Suit Winner task, we consider an image label $z \in \{\text{"Two of Hearts"}, \text{"Three of Hearts"}, \dots, \text{"Ace of Clubs"}\}$, denoting the suit and rank value of one of 52 playing cards. The question-answer generator takes z as input, and creates two entries in the lookup-table. For example, if $z = \text{"Jack of Hearts"}$:

$$\begin{aligned} l[\text{"which rank is this?"}, \text{"Jack of Hearts"}] &= \text{"Jack"} \\ l[\text{"which suit is this?"}, \text{"Jack of Hearts"}] &= \text{"Hearts"} \end{aligned}$$

There are therefore two questions: $Q = \{\text{"which rank is this?"}, \text{"which suit is this?"}\}$, with possible answers $\mathbf{A}_{q_1} = \{\text{"Two"}, \text{"Three"}, \dots, \text{"Ace"}\}$, and $\mathbf{A}_{q_2} = \{\text{"Hearts"}, \dots, \text{"Clubs"}\}$ respectively. An example data point is shown in Figure 6.2a.

Plant Hitting Sets. In this task, we consider a modification of the Hitting Sets task studied in Chapter 4, where images of plant leaves with various diseases are used (Singh et al., 2020) instead of MNIST digits. The features of interest are the species and disease types of each

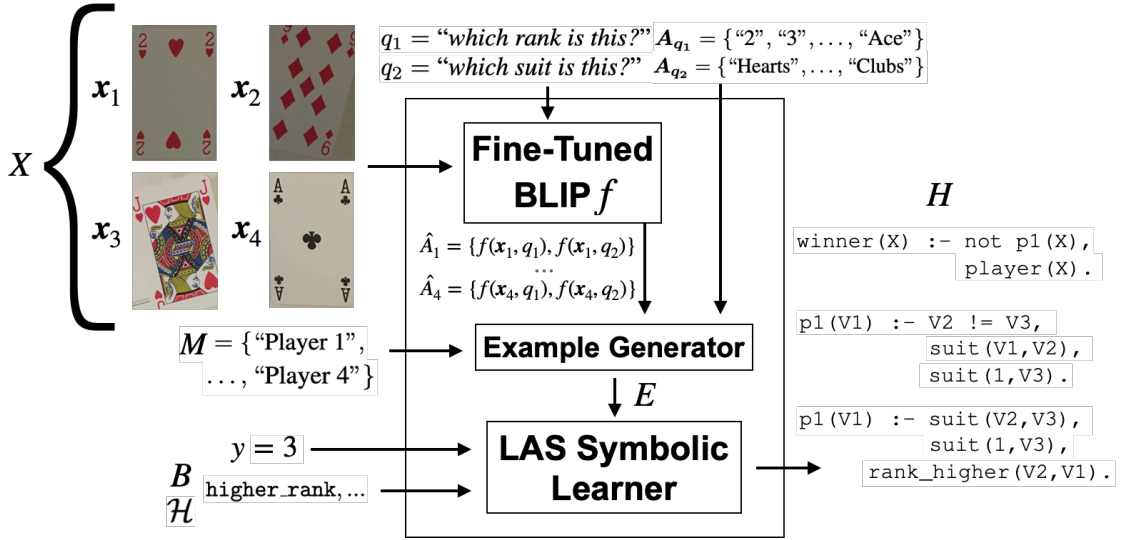
Input: Image x with label z .
Output (after observing all images): Fine-tuned BLIP model, a set of questions Q , and possible answers for each question $A_{q \in Q}$.



(a) BLIP Fine-Tuning

Input: Image sequence X , meta-data M , task label y , background knowledge B , hypothesis search space $\mathcal{H}$.

Output (after observing multiple examples): Optimal learned hypothesis H .



(b) Symbolic Rule Learning

Figure 6.2: NeSyGPT architecture with one data point in the Follow Suit task. The goal is to learn the rules of the game: The winner is the player with the highest ranked card with the same suit as Player 1. (a) BLIP is fine-tuned using playing card images and natural language questions and answers. (b) A hypothesis is learned from BLIP predictions. Note in (a), fine-tuning occurs with both suit and rank answers.

plant. Excluding the “background” class, there are 38 possible species and disease combinations, including a “healthy” class for each species.¹ The question-answer generator takes as input an image label $z \in \{1 \dots 38\}$, and creates two entries in the lookup table. For example, given the

¹The exceptions are the Orange and Squash species, which do not have a healthy class.

image label $z = 1$, which represents the Apple species with the Scab disease, the lookup table contains:

$$\begin{aligned} l[\text{"which species is this?"}, 1] &= \text{"Apple"} \\ l[\text{"which disease is this?"}, 1] &= \text{"Scab"} \end{aligned}$$

In this case, $Q = \{\text{"which species is this?"}, \text{"which disease is this?"}\}$, and $\mathbf{A}_{q_1} = \{\text{"Apple"}, \dots, \text{"Tomato"}\}$, and $\mathbf{A}_{q_2} = \{\text{"Black rot"}, \dots, \text{"Yellow Leaf Curl Virus"}\}$ respectively.

CLEVR-Hans. Finally, we demonstrate NeSyGPT can solve tasks that require detecting various properties of objects within an image (i.e., it can extend beyond image classification), using the CLEVR-Hans3 task. Each input image contains up to 10 objects, and the features of interest are the *size*, *material*, *colour*, and *shape* of the different objects. In this task, the image label z is a natural language description of the object properties within each image. For example, $z = \text{"The image contains a large gray cube made of metal, a large yellow cylinder made of metal, a small brown cube made of metal, ..."}.$ To fine-tune BLIP in this case, we simply use one question $Q = \{\text{"what does the image contain?"}\}$, and fine-tune BLIP to output the full natural language description of the various objects and their properties within an image. The lookup-table therefore returns the image label itself, i.e., for a given z :

$$l[\text{"what does the image contain?"}, z] = z$$

6.2.2 Learning a Hypothesis

Example Generator. Given a task data point $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$, and a sequence of answer meta-data pairs $\langle (\hat{A}_1, m_1), \dots, (\hat{A}_n, m_n) \rangle$, where the answers are predicted from the fine-tuned BLIP model, the goal is to generate a LAS training example $e \in E$ for the symbolic learner. The inclusion and exclusion sets of e encode the downstream label. Formally, $e_{\text{inc}} = \{y\}$, and $e_{\text{exc}} = \{y' : y' \in \mathcal{Y}, y' \neq y\}$. The answers and meta-data are encoded into the example's context e_{ctx} . In the case of image classification, (e.g., *"which digit is this?"*), there is no guarantee that BLIP predicts a valid answer for a given question. To tackle this problem, we use the Levenshtein string distance to return the answer $a \in \mathbf{A}_q$ with the lowest edit distance compared to the predicted answer (Yujian and Bo, 2007). The answers and meta-data are then encoded into e_{ctx} in ASP form, which is task specific. Also, for simplicity, we apply a uniform weight penalty across all examples, since this performed similarly to a confidence-based weight penalty in Chapter 3, and confidence information is not readily available from the output of BLIP. This takes the value 10 in all experiments, except CLEVR-Hans, where a penalty of 1 is used. 10 and 1 are chosen because they provide adequate balance against the length of the hypothesis. In CLEVR-Hans, since each example represents a highly specific case, more examples are required. Therefore, a lower weight penalty is applied to each example, in order to bias the symbolic learner towards learning a shorter, more general hypothesis. The Example Generator can be manually engineered, or obtained automatically using a LLM, which generates a Python function that encodes the ASP representation of a full training example e . In our evaluation, we demonstrate that with small syntactic fixes, the LLM can encode the training examples with competitive accuracy compared to a manual specification. Let us now present instantiations of the Example Generators used in each of our evaluation tasks.

Two-Digit Arithmetic. In these tasks, BLIP is fine-tuned with the question $q_1 = \text{"which digit is this?"}$, and the possible answers are $\mathbf{A}_{q_1} = \{0 \dots 9\}$. Let us assume an $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data

```

def generate_example(
    example_id: int,
    blip_predictions: list[dict[str, str]],
    label: str,
    meta_data: list | None = None
) -> WCDPI:
    """
    This function generates a WCDPI for the Two-Digit Arithmetic task.
    Args:
        example_id (int): The identifier for the example.
        blip_predictions (list[dict[str, str]]): A list of predictions from BLIP. The list contains
            a dictionary of questions and answers related to the concepts in each image.
        label (str): The target task label.
        meta_data (list, optional): The optional meta-data associated with the data point.
    Returns (WCDPI): A WCDPI object representing the training example for the symbolic
        learner.
    """
    # Create the example context using the BLIP predictions for each image.
    context: list[str] = []
    for image_index, image_predictions in enumerate(blip_predictions):
        digit = image_predictions["which digit is this?"]
        context.append(f"digit({image_index + 1},{digit}).")

    # Construct the WCDPI object.
    return WCDPI(
        ex_id=f"ex_{example_id}",
        positive=True,
        weight=10,
        inclusion=[f"label({label})"],
        exclusion=[],
        context=context
    )

```

Listing 6.1: Python implementation of the Example Generator for the Two-Digit Arithmetic task.

point for the Addition task, where $X = \langle \text{2}, \text{1} \rangle$, $M = \langle \rangle$, and $y = 3$. Let us also assume BLIP is fine-tuned to perfect accuracy. The sequences of answers are: $\hat{A}_1 = \langle \text{"2"} \rangle$, and $\hat{A}_2 = \langle \text{"1"} \rangle$. The generated example for the symbolic learner is therefore:

```

#pos(id@10, { result(3) }, { }, {
    digit(1,2).
    digit(2,1).
}).

```

As in previous chapters, B includes a constraint to ensure only one label can be output for a given pair of digits, so we omit the other labels from the exclusion set. The above example can be generated using the python function shown in Listing 6.1, which forms the Example Generator for this task.

Follow Suit Winner. BLIP is fine-tuned with the questions $Q = \{ \text{"which rank is this?"}, \text{"which suit is this?"} \}$, and the answers $A_{q_1} = \{ \text{"Two"}, \dots \text{"Ace"} \}$, and $A_{q_2} = \{ \text{"Hearts"}, \dots, \text{"Clubs"} \}$ respectively. Let us assume BLIP predicts correctly for the $\langle X, M, y \rangle \in \mathcal{D}_{\text{task}}$ data point presented in Figure 6.2b. The sequences of predicted answers are: $\hat{A}_1 = \langle \text{"2"}, \text{"Hearts"} \rangle$, $\hat{A}_2 = \langle \text{"9"}, \text{"Diamonds"} \rangle$, $\hat{A}_3 = \langle \text{"Jack"}, \text{"Hearts"} \rangle$, and $\hat{A}_4 = \langle \text{"Ace"}, \text{"Clubs"} \rangle$. The Example Generator is similar to the Example Generator for the Two-Digit Arithmetic task shown in List-

ing 6.1, except the contextual facts are constructed using the suit and rank questions. Also, the meta-data denoting the player of each card is encoded alongside the suit and rank predictions in a `card` predicate. The full python code listing of the Example Generator is given in Appendix B. For the data point shown in Figure 6.2b, the Example Generator outputs:

```
#pos(id@10, { winner(3) }, { winner(1), winner(2), winner(4) }, {
    card(1,2,h).
    card(2,9,d).
    card(3,j,h).
    card(4,a,c).
}).
```

Plant Hitting Sets. BLIP is fine-tuned using the questions $Q = \{ \text{“which species is this?”}, \text{“which disease is this?”} \}$, and the answers $\mathbf{A}_{q_1} = \{ \text{“Apple”, “Blueberry”, } \dots, \text{“Tomato”} \}$, and $\mathbf{A}_{q_2} = \{ \text{“Black rot”, } \dots, \text{“Yellow Leaf Curl Virus”} \}$ respectively. Given the predictions from BLIP, we transform the species and disease answers into a class identifier in the range $\{1 \dots 38\}$, based on the mapping given in the Plant Disease dataset (Singh et al., 2020). Note in this task, we apply the Levenshtein string distance comparison to both species and disease answers independently, comparing to all possible species, and all possible diseases, respectively. Since certain diseases can only occur with a certain species, occasionally an invalid species and disease combination is predicted, and therefore there is no corresponding class identifier. In these cases, we choose a random valid disease for the predicted species, and then obtain a valid class identifier. This occurs very rarely in our experiments. The examples for the symbolic learner are structured as follows:

```
#pos(id@10, { }, { }, {
    ss_element(1,35).
    ss_element(1,4).
    ss_element(2,19).
    ss_element(2,14).
    ss_element(2,35).
    ss_element(3,10).
}).
```

where the first argument of the `ss_element` predicate is the set identifier, and the second argument is the plant image class identifier. The target label denotes whether the collection of sets contains a hitting set or not, which is encoded via positive and negative examples. The full python code listing for the Example Generator for this task is shown in Appendix B.

CLEVR-Hans. BLIP is fine-tuned with $Q = \{ \text{“what does the image contain?”} \}$, and a ground-truth natural language description of the objects in the image as the answer. As an example, this takes the form *“The image contains a large gray cube made of metal, a large yellow cylinder made of metal...”*. During inference, we parse the output sentence from BLIP and extract the various properties associated with each object. We then create an example for the symbolic learner as follows:

```
#pos(id@1, { label(0) }, { label(1), label(2) }, {
    size(1, large).
    color(1, gray).
    shape(1, cube).
    material(1, metal).
    size(2, large).
    color(2, yellow).
    shape(2, cylinder).
    material(2, metal).
    ...
}).
```

where the object properties are encoded into the example’s context, and the target label is encoded into the inclusion and exclusion sets. Recall in this task, we use a uniform weight of 1, to balance adequately against the length of the hypothesis, since more training examples are given than in our other evaluation tasks. The full python code listing for the Example Generator for this task is given in Appendix B.

In all evaluation tasks, given a set of training examples E , and a domain knowledge B , the symbolic learner learns an optimal hypothesis H , satisfying Equation 2.27. H , together with B , forms the reasoning function $h = H \cup B$. Let us now present our experimental results.

6.3 Evaluation

Our evaluation investigates whether the implicit knowledge embedded within BLIP can improve the accuracy of neuro-symbolic learning and reasoning, whilst reducing the amount of labelled data required. We compare NeSyGPT against the following types of architectures: *Neuro-Symbolic Reasoning*, where the symbolic hypothesis is given, *Neuro-Symbolic Learning*, where the hypothesis is learned, and *fully neural methods*, including deep learning, and generative models.

Our four problem domains, namely, Two-Digit Arithmetic, Follow Suit Winner, Plant Hitting Sets, and CLEVR-Hans, each have their own characteristics. The Two-Digit Arithmetic domain includes the MNIST Addition task from (Manhaeve et al., 2018), and the E9P task from Chapter 4, where we demonstrate learning negation as failure. The purpose of these tasks is to provide a comparison to other systems on a standard benchmark. The Follow Suit Winner domain from Chapter 3, tests scalability with respect to the number of concept values. In this chapter, we introduce a 10-player variant, which significantly increases the number of possible values. A given data point has $\sim 6.5\text{m}$ possible values in the 4-player variant, and $\sim 5.74 \times 10^{16}$ possible values in the 10-player variant. For reference, the 2-digit MNIST Addition task has 100 possible values per data point. This domain also requires learning predicates that are not directly observed in the examples (i.e., predicate invention). The Plant Hitting Sets domain includes real-world images, and requires learning complex rules including constraints and choice, as well as reasoning over multiple models (answer sets). The purpose of this task is to demonstrate practical utility on a task closer to a real-world problem domain. Finally, the CLEVR-Hans task tests more advanced perception ability, i.e., object detection, since up to 10 objects and their properties need to be extracted from a single image. This is significantly harder than extracting concept values via image classification.

The results show that NeSyGPT: (1) Achieves state-of-the-art performance compared to the other approaches, when learning in such low-label settings. (2) Can scale to complex tasks with a large number of choices for the concept values. (3) Can easily integrate symbolic learning of ASP programs to learn a complex and interpretable hypothesis. (4) Can achieve state-of-the-art performance on CLEVR-Hans, when given the same number of labels as the next best approach. We also demonstrate that the amount of manual engineering can be reduced using GPT4 (OpenAI et al., 2023), as GPT4 can generate reasonable questions and answers for fine-tuning BLIP, and for generating examples for the symbolic learner.

6.3.1 Experimental Setup

Baselines. Our baselines have symbolic components that are either fully neural, or are capable of learning or representing complex hypotheses expressed in ASP. The motivation is to demon-

strate that the proposed benefits of NeSyGPT are due to the implicit knowledge embedded within BLIP, and our novel method of integrating BLIP with the symbolic learning component, as opposed to the symbolic representation used. We choose different types of baselines: (1) Neuro-Symbolic Reasoning, where the symbolic knowledge is given, (2) Neuro-Symbolic Learning, where the symbolic knowledge is learned, and (3) Fully neural methods.

For the Neuro-Symbolic Reasoning baselines, we compare our approach to **NeurASP** (Yang et al., 2020), **SLASH** (Skryagin et al., 2023), and two variants of Embed2Sym (Aspis et al., 2022), one trained end-to-end as usual (**Embed2Sym**), and another where clustering is performed on an embedding space resulting from fine-tuning a perception model with ground-truth image labels (**Embed2Sym***). This more closely reflects our method of fine-tuning, whilst obtaining concept values via a clustered embedding space, and an ASP optimisation. We also use a modified version of ABL (Dai et al., 2019) as an additional baseline, which uses the same perception component as our approach, and an ASP implementation of their abduction step (**BLIP+ABL**). Note that BLIP cannot directly be used in NeurASP, SLASH, and Embed2Sym because these methods require back-propagating gradients to a softmax layer for training the perception component. BLIP on the other hand, is fine-tuned with natural language questions and answers. ABL identifies exact pseudo-labels for training the perception component, and therefore can be used to generate pseudo-answers for fine-tuning BLIP. For NeurASP, SLASH, and Embed2Sym, we use the best pre-trained model available; the Vision Transformer *ViT-B-32* from PyTorch, which is pre-trained on ImageNet.²

For the Neuro-Symbolic Learning baselines, we compare to **FF-NSL** from Chapter 3 (Cunnington et al., 2023a) and **NSIL** from Chapter 4 (Cunnington et al., 2023b), using the pre-trained ViT-B-32 as the perception model. For fully neural baselines, we compare to: (1) The ViT-B-32 connected to a fully differentiable reasoning network (**ViT+ReasoningNet**), which is a MLP in the Two-Digit Arithmetic and Follow Suit Winner domains, and a Recurrent Transformer in the Plant Hitting Sets domain, since longer sequences are observed (Hutchins et al., 2022). (2) BLIP connected to GPT4 for rule learning (**BLIP+GPT4**). We prompt GPT4 to generate Python programs for the Two-Digit Arithmetic and Follow Suit domains, since it is easy to express solutions to these tasks in Python, and GPT4 has likely seen many Python examples during pre-training. For the Plant Hitting Sets domain, we prompt GPT4 to generate ASP programs, since it is more natural to express the solution to this problem in a declarative manner. (3) BLIP solving the downstream task with all images contained within a single image (**BLIP Single Image**). Finally, in the CLEVR-Hans domain, we consider **α ILP** as our baseline (Shindo et al., 2023), since α ILP has achieved state-of-the-art performance on this task.

Training data. In the Two-Digit Arithmetic, Follow Suit, and Plant Hitting Sets tasks, we generate questions and answers for fine-tuning BLIP using a varying number of image labels: $\{0, 1, 2, 5, 10, 20, 50, 100\} \times$ the number of concept values for each image. To learn the symbolic hypothesis, we use $\{10, 20, 50, 100\}$ labelled examples for the symbolic learner. In all datasets, we use an even distribution of labels. For the baselines, we use the same total number of labels, distributed as follows. For NeurASP, SLASH, Embed2Sym, BLIP+ABL, NSIL, and BLIP Single Image, we train these models end-to-end with the same total number of labels as required for our approach, with no supervision on the concept values. For Embed2Sym*, FF-NSL, ViT+ReasoningNet, and BLIP+GPT4, we follow a similar approach to ours: We first fine-tune the perception component using ground-truth image labels, and then train the reasoning component using the same number of task labels as we used for training the symbolic learner. Note

²<https://pytorch.org/vision/stable/models.html>

that for Embed2Sym*, the reasoning component is the ASP cluster labelling stage, whereas for FF-NSL, ViT+ReasoningNet, and BLIP+GPT4, the knowledge for the reasoning component is learned, using labelled examples with image predictions from the fine-tuned perception component. In the CLEVR-Hans domain, we perform two variations of BLIP fine-tuning; (1) With 70k labelled images from the original *CLEVR* dataset to enable a fair comparison to α ILP, and to demonstrate data efficiency, (2) with 9000 labelled images from the *CLEVR-Hans* training set.

Symbolic Learner. In the Two-Digit Arithmetic and CLEVR-Hans domains, we use FastLAS for the symbolic learner due to FastLAS’ increased efficiency in larger search spaces for this specific class of problems (Law et al., 2020a). We use ILASP in the Follow Suit Winner and Plant Hitting Sets domains, as ILASP supports predicate invention (required in the Follow Suit Winner domain), and the ability to learn choice rules and constraints (required in the Plant Hitting Sets domain) (Law et al., 2020b). Recall we set the LAS example weight penalty to 10 in all experiments except CLEVR-Hans, where a penalty of 1 is used. 10 and 1 are chosen because they provide adequate balance against the length of the hypothesis. In CLEVR-Hans, there are more symbolic examples given, so a lower weight penalty is required to learn a shorter, more general hypothesis.

General Configuration. Each experiment is repeated 5 times using 5 randomly generated seeds and datasets. We use the “vqav2” BLIP model from the LAVIS library (Li et al., 2022a), which is pre-trained on the Visual Genome, Conceptual Captions, Conceptual 12M, SBU captions, LAION, and the VQA v2.0 datasets. In this work, BLIP is fine-tuned using a distinct set of training images, which to the best of our knowledge, do not occur in the datasets used for pre-training BLIP. Also, the images in the training examples for the symbolic learner are not used for BLIP fine-tuning in our experiments. We use 5 held-out test sets (not seen in any training stage) to evaluate task accuracy, and report the mean and standard error. For all models, we train with the most performant number of epochs we can achieve within a 24-hour time limit. We train NeSyGPT for 10 epochs in the Two-Digit Arithmetic and Follow Suit Winner domains, and 20 epochs on the Plant Hitting Sets and CLEVR domains. For all baselines, we train for 50 epochs unless otherwise stated, as a reasonable upper-bound to ensure convergence. All hyper-parameters are left as default, and we use the SGD optimiser with a learning rate of 0.01 for all models, unless specified in a methods codebase. All experiments are run on a shared cluster with the following specification: RHEL 8.5 x86 64 with Intel Xeon E5-2667 CPUs (10 cores total), and 1 NVIDIA A100 GPU, 150GB RAM. The ASP listings of the domain knowledge and search space used in each experiment are given in Appendix A.

Specific Baseline Configuration. Some baselines required additional configuration, and occasionally, modifications, in order to obtain results. Let us now present these details.

- **NeurASP.** In the Follow Suit Winner and Plant Hitting Sets domains, we limit the number of epochs to 10, and the number of answer sets returned for each training example to 5000, in order to obtain results within the 24-hour time limit.
- **SLASH.** We use the SGD optimiser with a learning rate of 0.01, as we obtain “nan” values when calculating the gradients with the default Adam optimiser given in the SLASH codebase. This matches the other baselines.
- **BLIP+ABL.** To match the NeSyGPT results, we train BLIP+ABL for 10 iterations in the Two-Digit Arithmetic and Follow Suit Winner domains, and 20 iterations on the Plant Hitting Sets domain. For Two-Digit Arithmetic, we fine-tune BLIP for 10 epochs with

a given set of abducted pseudo-labels on each iteration. For the Follow Suit Winner and Plant Hitting Sets domains, we restrict the number of epochs to 1, but this did not prevent a timeout.

- **NSIL.** We train the perception network for 1 epoch given the learned hypothesis at each iteration. We also limit the number of answer sets to 5000 for each training data point, as in the Follow Suit domain, there are too many corrective examples for ILASP. This is also required to train the network in NeurASP. Furthermore, we restrict NSIL to 10 iterations, in order to avoid a timeout. We also restrict the number of corrective examples by removing examples with weights less than 5 (the lower the weight, the less influence the example has on the optimisation of the symbolic learner). Finally, we fail to obtain NSIL results in the Plant Hitting Sets domain, because the implementation does not support variable length training data points, and when attempting to run the initial bootstrap stage manually, the symbolic learner fails to return a hypothesis within a few hours. As NeurASP also times out in this domain, we can assume NSIL would also reach a timeout, even if the bootstrap task were to succeed.
- **ViT+ReasoningNet.** We experiment using a MLP, a LSTM and a Recurrent Transformer as the reasoning network in the Follow Suit Winner domain, and conclude the MLP has superior performance.
- **BLIP+GPT4.** We use the same BLIP perception model as NeSyGPT, and prompt GPT4 to generate a program to solve the reasoning task. We do not include the target hypothesis directly in the prompt, but we do explain the various relations that can be used, to simulate defining the hypothesis search space, as occurs in the symbolic learner of NeSyGPT. We use the “gpt-4” model from the OpenAI API, perform 5 repeats with 5 random seeds, and set the max tokens parameter to 5000. We access the API between 20th November — 30th December 2023. We present the prompts used for each domain in the relevant sub-sections.
- **BLIP Single Image.** For the Follow Suit Winner domain, we include player annotations for each image, and for the Plant Hitting Sets domain, we encode the set assignment information as brackets within the image. All images are square as required by the BLIP image processors. We show example data points for each domain in the relevant sub-sections.

6.3.2 Two-Digit Arithmetic

This domain includes the Addition task from (Manhaeve et al., 2018), and the E9P task from Chapter 4, where the input is 2 MNIST images, and the label is equal to the digit value of the second image, if the digit value of the first image is even, or 9 plus the digit value in the second image otherwise. The search space for the symbolic learner follows our evaluation in Chapter 4, and includes the relations `even`, `not even`, `plus nine`, and `=`, as well as the function `+`. Image accuracies are reported over the standard MNIST test set of 10,000 images, and task accuracies are reported over held-out test sets of 1000 examples. For the BLIP Single Image baseline, we fine-tune BLIP using a concatenated pair of MNIST digit images, and the downstream label:

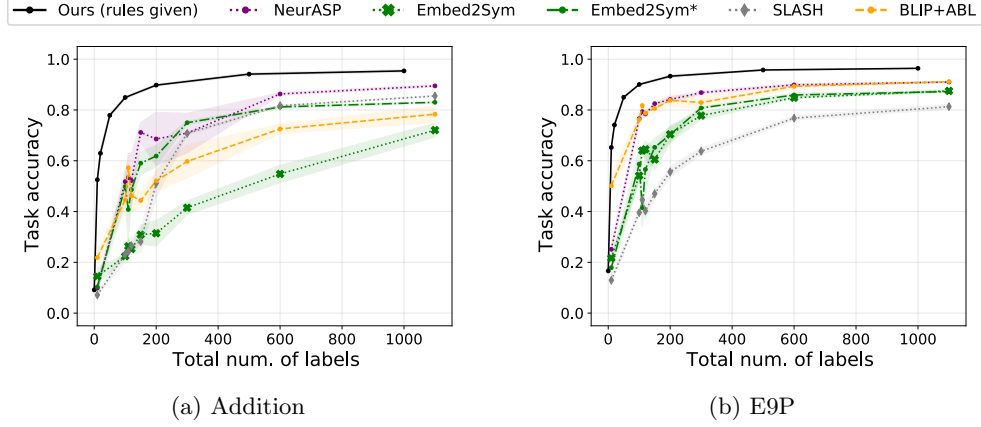


Figure 6.4: Two-Digit Arithmetic task accuracy when the hypotheses are given.

Figure 6.3: Example data point for the BLIP Single Image baseline. in the case of Addition, BLIP is fine-tuned to output $y = 8$.

For the BLIP+GPT4 baseline, the prompt for GPT4 is shown in Listing 6.2, where the example input contains predictions from BLIP.

```
The task is to generate a python program called "compute_answer" that returns a target answer, given a pair of
digits as input. The program should generalise across a set of training examples and output the correct
answer for all of the given input digit pairs. The digits are represented using a list, where the first
and second elements denote the values of the first and second digit respectively. Note that the digits
could be noisy as they are output from neural network predictions. The training examples are:

Example 1:
Input: [4, 8]
Output: 12

Example 2:
Input: [4, 6]
Output: 10

<<Insert more examples here>>

As background knowledge, consider using code that computes; (1) whether a given digit is even or not, (2) the
arithmetic sum "9 + the given digit", (3) arithmetic addition, and (4) equality. Note that the python
function must generate exactly one output for a given pair of digits. The range of possible values for the
answer is 0-18, and for the input digits is 0-9, both inclusive. Generate the "compute_answer" python
program that covers as many of the training examples as possible, whilst minimising the length of the
program. Return your response as #### <python program> ####.
```

Listing 6.2: BLIP+GPT4 Prompt, Two-Digit Arithmetic tasks.

The results are shown in Figure 6.4 where the symbolic hypotheses are given, and Figure 6.5 where the hypotheses are learned. NeSyGPT outperforms all baselines, and learns the correct hypotheses (see Figure 6.6). In both cases, BLIP is fine-tuned to a mean image accuracy of 0.9216 with 100 labels, increasing to 0.9753 with 1000 labels. This is within 2.34% of the state-of-the-art result of 0.9987 (Byerly et al., 2021), whilst using only 1.67% of the 60,000 labels available. NeSyGPT is able to learn the correct hypotheses in all 5 repeats, when BLIP is fine-tuned using only 10 image labels (single-shot, one label per digit). To do so, the symbolic learner requires 50 task labels (examples) for the Addition task, and 20 for the E9P task. Note that

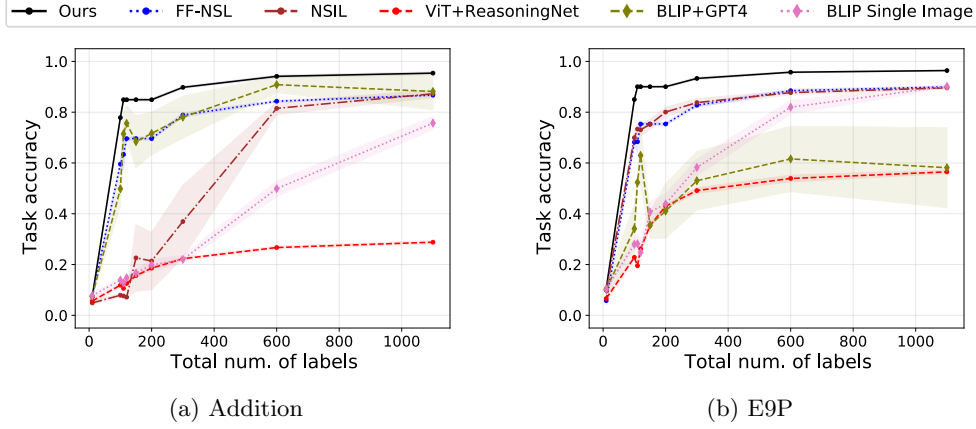


Figure 6.5: Two-Digit Arithmetic task accuracy when the hypotheses are learned.

Addition <code>result(V0,V1,V2) :- V2 = V0 + V1.</code> E9P <code>result(V0,V1,V2) :- even(V0), V2 = V1.</code> <code>result(V0,V1,V2) :- not even(V0), plus_nine(V1,V2).</code>

Figure 6.6: Two-Digit Arithmetic learned hypotheses

the NeSyGPT curves in Figure 6.4 are shifted to the left, because no examples are required for the symbolic learner, since the hypotheses are given. All baselines use the same total number of labelled data points as NeSyGPT does, when NeSyGPT learns the hypotheses (Figure 6.5).

In terms of the performance of the baselines in Figure 6.4, all methods except SLASH perform better on the E9P task than Addition. This is expected, as the downstream label is more informative in E9P: There is a reduced number of possible digit choices for certain examples, which gives a stronger signal for training the perception model. NeSyGPT is unaffected by this, given it is fine-tuned directly with answers generated from image labels. The end-to-end variant of Embed2Sym struggles to achieve a good clustering in the embedding space of the perception model in Addition, again because it is more difficult to distinguish the individual digits in these examples when training in an end-to-end fashion. Embed2Sym* on the other hand, achieves a better clustering, because the perception component is fine-tuned directly with concept labels. To illustrate this, Figure 6.7 shows the TSNE plots of the embedding spaces for the first repeat, when the models are trained with 1100 labels. For reference, Figure 6.7 shows the same plots but for the E9P task, where the clustering is similar between the two variants. This explains the performance difference in Figure 6.4a, and the similar performance in Figure 6.4b of the two variants.

In Figure 6.5, all methods except BLIP+GPT4 also perform better on the E9P task compared to Addition. BLIP+GPT4 fails to generate the correct hypothesis in the E9P task. The E9P task is less common than Addition, and it is unlikely GPT4 has seen many examples of this during pre-training. The fully neural ViT+ReasoningNet struggles in such low-label settings, whereas the neuro-symbolic approaches perform better. NSIL has to explore more of the search space in Addition before converging to the correct hypothesis, and FF-NSL follows a similar curve to NeSyGPT, but has lower performance due to an inferior perception component. Finally, BLIP Single Image approaches the performance of NeSyGPT, but requires a larger number of labelled data points to do so.

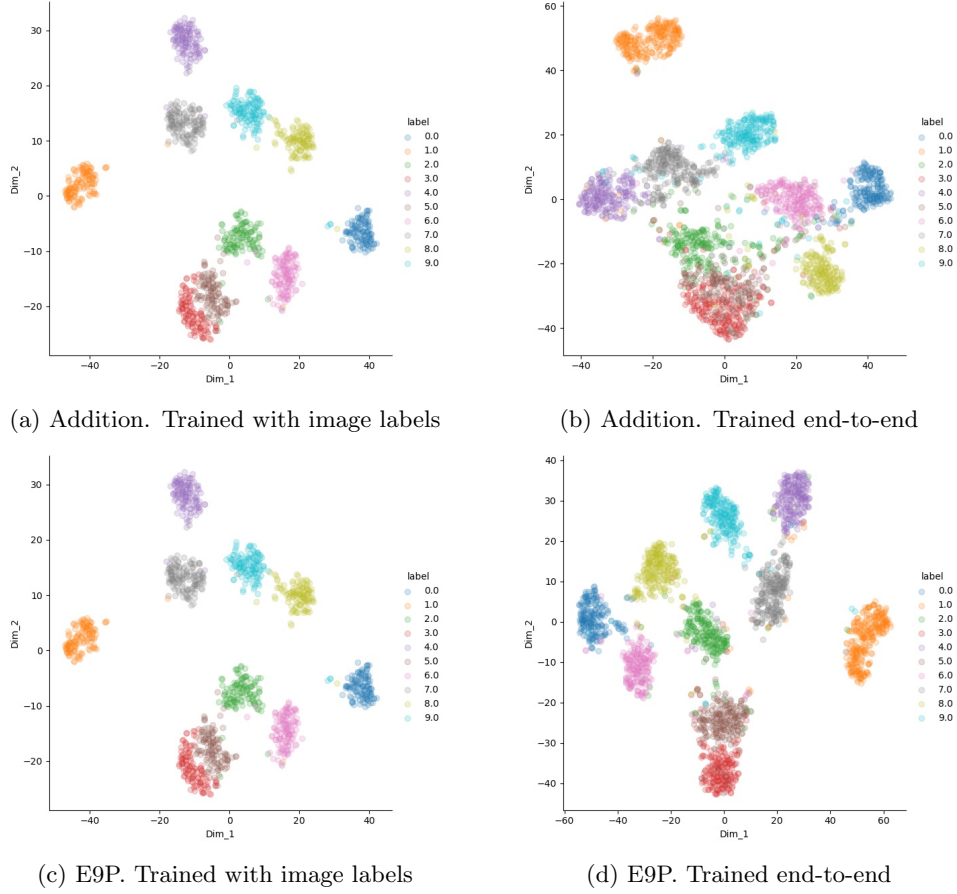


Figure 6.7: Embedding space comparison between the Embed2Sym variants on the Two-Digit Arithmetic tasks. (Left) Embed2sym*. (Right) Embed2Sym. (Top) Addition. (Bottom) E9P.

6.3.3 Follow Suit Winner

The Follow Suit Winner domain requires learning the rules of a trick-based playing card game, introduced in Chapter 3. Recall that each player plays a card from a standard deck of 52 cards, and the winner is the player with the highest ranked card with the same suit as the first player. This domain involves a large amount of choice for the values of the concepts, and therefore presents a challenge for end-to-end methods. To investigate this, we introduce a new 10-player variant, which significantly increases the number of choices compared to the 4-player variant. As explained during the introduction to our evaluation, assuming a single deck of cards, the 4-player variant has $\sim 6.5\text{m}$ choices for the values of the concepts, whereas the 10-player variant has $\sim 5.74 \times 10^{16}$ choices. The search space for the symbolic learner follows Chapter 3, and includes the relations `higher_rank` of arity 2, which denotes a certain player having a higher ranked card than another player, and the `!=` relation to denote two suit values are not equal. The search space also includes an instruction to perform predicate invention, to enable the definition of a winning player to be learned. The suit and rank values of each playing card are given as background knowledge, and the full background knowledge listing is presented in Appendix A.

For images, we use the *Standard* and *Adversarial Captain America (ACA)* decks from Chapter 3, where the ACA deck contains card images from a Captain America card deck placed on a background of standard card images. This is the most challenging deck from those considered in Chapter 3. BLIP is fine-tuned with the questions “*which rank is this?*”, and “*which suit is*

this?”. Image accuracies are reported over a test set of 1040 card images, and task accuracies are reported over held-out test sets of 1000 examples. For the BLIP Single Image baseline, we fine-tune BLIP using a concatenated set of playing card images, and the downstream label:



Figure 6.8: Example data point for the BLIP Single Image baseline. in this case, BLIP is fine-tuned to output $y = 3$.

For the BLIP+GPT4 baseline, the prompt for GPT4 is shown in Listing 6.3, where the example input contains predictions from BLIP.

```
The task is to generate a python program called "compute_answer" that returns a target answer for a card game,
given a list of 4 playing cards as input. The cards in the list correspond to a card played by each of 4
players, and the answer denotes the winning player in the card game. Each player's card is denoted by the
rank and then the suit, for example "kh" represents the card king of hearts, and "as" represents the card
ace of spades. Note that for "10d", the rank is 10 and the suit is diamonds. The program should generalise
across a set of training examples and output the correct answer for all of the given examples. Note that
the cards could be noisy as they are output from neural network predictions. The training examples are:

Example 1:
Input: ['8s', '9d', '10s', '7d']
Output: 3

Example 2:
Input: ['8c', 'js', '10h', '6c']
Output: 1

<<Insert more examples here>>

As background knowledge, consider using code that computes; (1) whether the suit of one player's card is equal
to a suit of another player's card, and (2), whether the rank of one player's card is higher than the rank
of another player's card. Note that the python function must generate exactly one output for a given list
of cards. Consider the following python function that computes the value of a playing card rank, given as
input a playing card. This can be used in the program:

def compute_rank_value(card):
    rank = card[:-1]
    if rank == 'j':
        rank_value = 11
    elif rank == 'q':
        rank_value = 12
    elif rank == 'k':
        rank_value = 13
    elif rank == 'a':
        rank_value = 14
    else:
        rank_value = int(rank)
    return rank_value

The range of possible values for the answer is 1-4, and assume the card inputs can be one of 52 playing cards
from a standard deck. Generate the "compute_answer" python program that covers as many of the training
examples as possible, whilst minimising the length of the program. Return your response as #### <python
program> ####.
```

Listing 6.3: BLIP+GPT4 Prompt, Follow Suit Winner tasks.

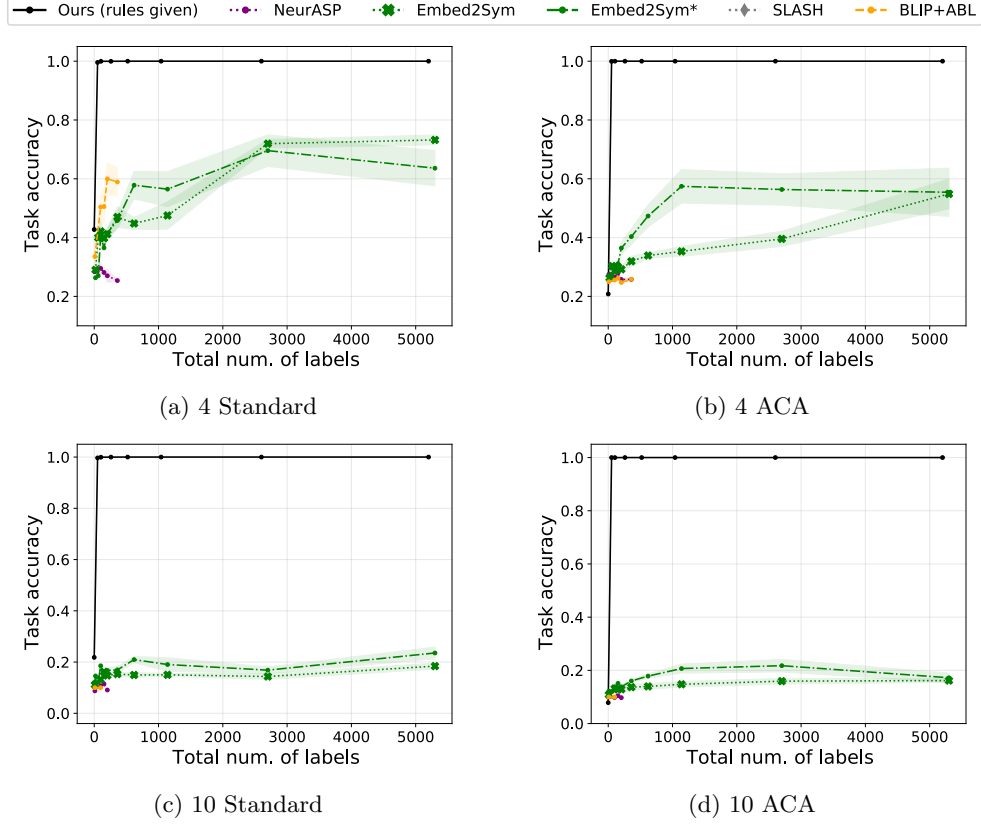


Figure 6.9: Follow Suit Winner task accuracy when the symbolic rules are given.

The results are shown in Figures 6.9 and 6.10 for reasoning and learning respectively. In both reasoning and learning, NeSyGPT achieves perfect performance on all variants, and significantly outperforms the baselines. As expected, all methods perform better on the 4-player variants than the 10-player variants, as the 10-player variant contains significantly more choice for the values of the concepts. When the hypothesis is given (Figure 6.9), NeurASP, SLASH, and BLIP+ABL all reach the 24-hour time limit, even with the 4-player standard variant. NeurASP struggles with the large number of choices, and since we limit the number of answer sets calculated during each example to 5000 (s.t. we can obtain results), NeurASP is not able to update gradients based on all the answer sets. We also had to limit NeurASP to 10 training epochs. SLASH became stuck in the ASP grounding stage, and therefore we are unable to obtain any results. For BLIP+ABL, the 10-player variant takes longer because there are more images in each example to calculate pseudo-labels for. Embed2Sym* is able to learn a good playing card clustering in the embedding space, but the ASP Optimisation for labelling the clusters fails to complete in full. We limit the ASP solver to return the best cluster labelling discovered within a 60 second timeout. For reference, the cluster labelling in the Two-Digit Arithmetic domain completed in less than a second. The end-to-end variant of Embed2Sym also struggles with cluster labelling, and is unable to learn a good clustering in the embedding space. The TSNE plots for the embedding spaces for the first repeat of the 4-player standard variant, when the models are trained with 5300 examples, are shown in Figure 6.12. You can see a clear improvement in clustering when the Embed2Sym* variant is used, compared to the end-to-end variant.

When learning the hypothesis, FF-NSL is the next-best baseline, but requires more labels to fine-tune the perception component to the same level of accuracy as NeSyGPT. As NSIL is based upon NeurASP, it suffers similar timeout issues. The fully neural VIT+ReasoningNet

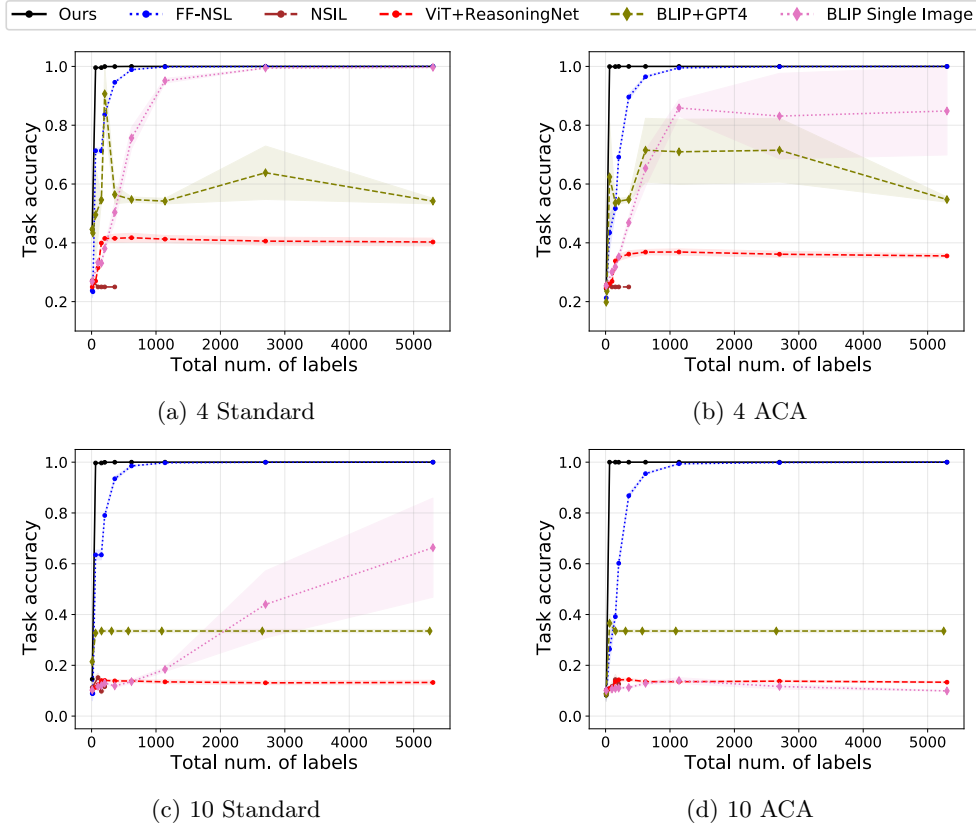


Figure 6.10: Follow Suit Winner task accuracy when the symbolic rules are learned.

```

winner(X) :- not p1(X), player(X).
p1(V1) :- V2 != V3; suit(1,V2); suit(V1,V3); player(V1); suit(V2); suit(V3).
p1(V1) :- higher_rank(V2,V1); suit(1,V3); suit(V2,V3); player(V1); player(V2); suit(V3).

```

Figure 6.11: Follow Suit Winner learned hypothesis

struggles in such low-label settings. BLIP+GPT4 and BLIP Single Image perform well in the 4-player variants, with BLIP Single Image achieving perfect accuracy given enough labels in the 4-player standard case. However, both of these approaches fail to solve the 10-player variant. For NeSyGPT, BLIP is fine-tuned to an image accuracy of 0.9832 and 0.9988 for standard and ACA decks respectively, with only 52 image labels (single-shot). This increases to 1.0 in all 5 repeats for both standard and ACA variants, given 520 and 2600 labels respectively. FF-NSL on the other hand, requires 5200 image labels to achieve perfect accuracy in all 5 repeats for both standard and ACA variants. For NeSyGPT, the correct hypothesis is learned with only 10 task labels, in all variants (see Figure 6.11).

6.3.4 Plant Hitting Sets

The goal is to learn a variation of the Hitting Set problem (Karp, 1972) from images of plant leaves of various crops (Singh et al., 2020). Let us assume a collection of crop fields, where each field contains a set of crops that can be diseased or healthy. In this context, a “hitting set” represents a set of diseased crops that intersects with each crop field. When the hitting set is calculated, this enables a prioritisation in terms of which diseases should be eradicated first, as one may wish to eradicate diseases that occur in each crop field. The size of the hitting set can be viewed as the budget for how many crops are able to be eradicated. The standard rules of

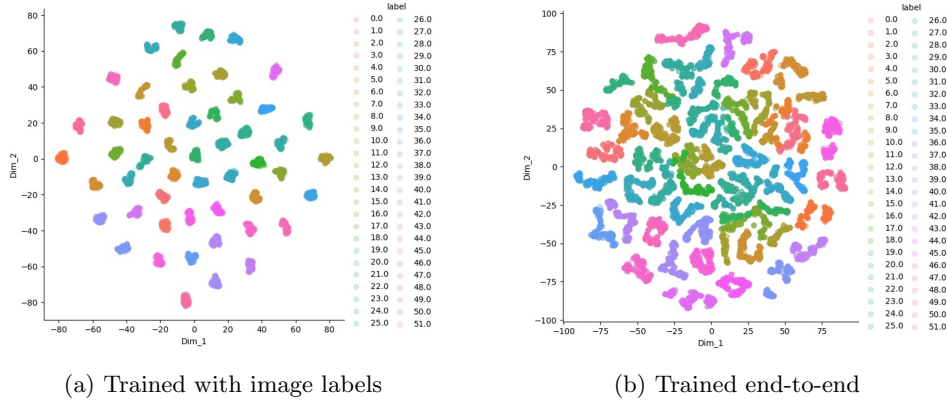


Figure 6.12: Embed2Sym embedding spaces — Follow Suit 4 Standard.

the Hitting Set problem apply, and we also require the constraint that no healthy crops should appear in the hitting set, since we are not interested in eradicating those. To solve this task, complex rules are required involving constraints, choice rules, and, to generate *all* the hitting sets with a certain budget, the solution requires multiple models.

The input consists of a collection of plant disease images arranged into various subsets. We generate the examples randomly, where each example contains at most 5 subsets, and each subset contains at most 5 elements. Note this is a significant extension to the evaluation presented in Chapter 4, since Chapter 4 contained various restrictions. For example, a maximum of 4 elements across the entire collection of subsets was assumed, and each subset was arranged into ascending order with no duplicate elements. In this chapter, we relax these restrictions, which significantly increases the difficulty of the task. An example data point is shown in Figure 6.13.

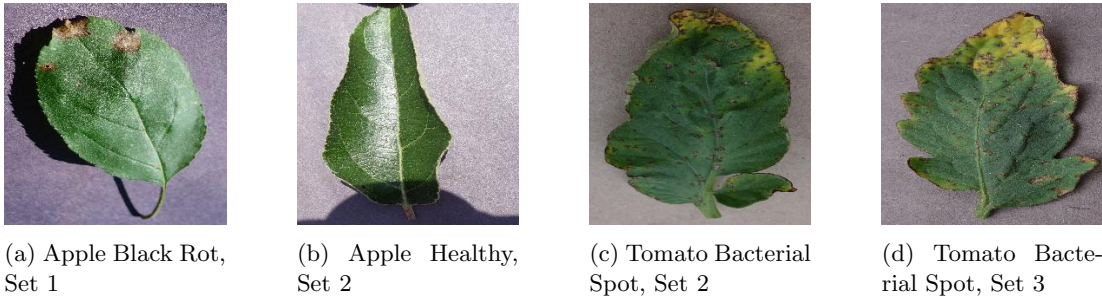


Figure 6.13: Plant Hitting Sets example data point. The meta-data M indicates which set each element belongs to. The label is $y = 1$ as there is a hitting set of size ≤ 2 with the elements Apple Black Rot, and Tomato Bacterial Spot.

In the plant disease image dataset (Singh et al., 2020), each element can take an integer value in $\{1..38\}$ to denote a particular plant species and disease. We remove the “background” class present in the dataset. We assume the hitting set size (budget) is 2, and the label is binary, indicating whether the collection contains a hitting set with no healthy images of size ≤ 2 . The search space for the symbolic learner contains the relation ‘!=’ and the relation **hs**, which defines an element of the hitting set. The budget 2, as well as the integer values which correspond to healthy plant images, are given as background knowledge. To fine-tune BLIP, we use the questions “*which plant species is this?*”, and “*which disease is this?*”. If the plant image is not diseased, we use the answer “*healthy*”. Image accuracies are reported over a test set of 5000 plant images, and task accuracies are reported over held-out test sets of 1000 examples.

To fine-tune the BLIP Single Image baseline, we create a single image to represent the entire collection of plant images arranged into subsets, and the objective is for BLIP to directly predict whether the collection contains a hitting set or not. For example, the following image contains a hitting set of size ≤ 2 , where no healthy crops occur in the hitting set:

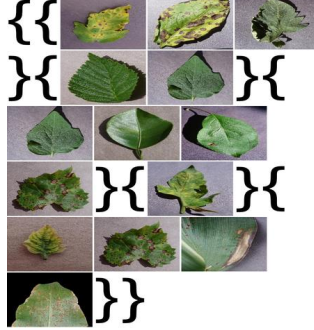


Figure 6.14: Example data point for the BLIP Single Image baseline. In this case, BLIP is fine-tuned to output $y = 1$, indicating this collection does contain a hitting set of size ≤ 2 , where the hitting set contains diseased crops only.

For the BLIP+GPT4 baseline, we prompt GPT4 to generate an ASP program, since it is more natural to express the solution to this task in a declarative manner. We also give substantial clues as to the correct rules, and provide context on the correct syntax of ASP. The full prompt is shown in Listing 6.4, where the element identifiers in the examples are predicted by BLIP.

```
The task is to generate an Answer Set Programming (ASP) program that decides whether a given collection of sets
contains a hitting set. A hitting set is a set that intersects with each set in the collection. Each set
contains integer values, which are given as input. In this task, the hitting set is restricted to be of
size 2 or less. In each example, elements in each set are denoted with the predicate "ss_el", where the
arguments correspond to the set identifier and the element itself. For example, "ss_el(1,5)." indicates
that set 1 contains the element 5. The program should generalise across a set of training examples and
return satisfiable or unsatisfiable (depending on whether the collection of elements contains a hitting
set) for all of the given examples. Note the "ss_el" predicates could be noisy as the elements are
generated from neural network predictions. The training examples are:

Example 1:
Input:
ss_el(1,24). ss_el(1,31). ss_el(1,33). ss_el(1,14). ss_el(1,3). ss_el(2,26). ss_el(2,6). ss_el(2,28). ss_el
(2,37). ss_el(2,31). ss_el(3,29). ss_el(3,18). ss_el(3,9). ss_el(4,10). ss_el(4,5). ss_el(4,24).
ss_el(4,26).
Has hitting set? No.

Example 2:
Input:
ss_el(1,36). ss_el(1,19). ss_el(1,31). ss_el(1,23). ss_el(2,19). ss_el(2,35). ss_el(2,7). ss_el(2,27).
ss_el(2,32).
Has hitting set? Yes.

<<Insert more examples here>>

The following ASP background knowledge should be used in
the program:

hs_index(1..2).
elt(1..38).
healthy(4). healthy(5). healthy(7). healthy(11). healthy(15). healthy(18). healthy(20). healthy(23).
healthy(24). healthy(25). healthy(28). healthy(38).

The "hs_index" predicate denotes the index in the hitting set (1 or 2). "elt" denotes the range of possible
integer elements. The "healthy" predicate indicates which integers have the healthy attribute and do not
appear in the hitting set. When generating the ASP program, consider using the following predicates in the
head of a rule: a "hs" predicate of arity 2 with the form "hs(X,Y)", which associates an element Y in the
hitting set with it is index in the hitting set X, and a "hit" predicate of arity 1 with the form "hit(X)
```

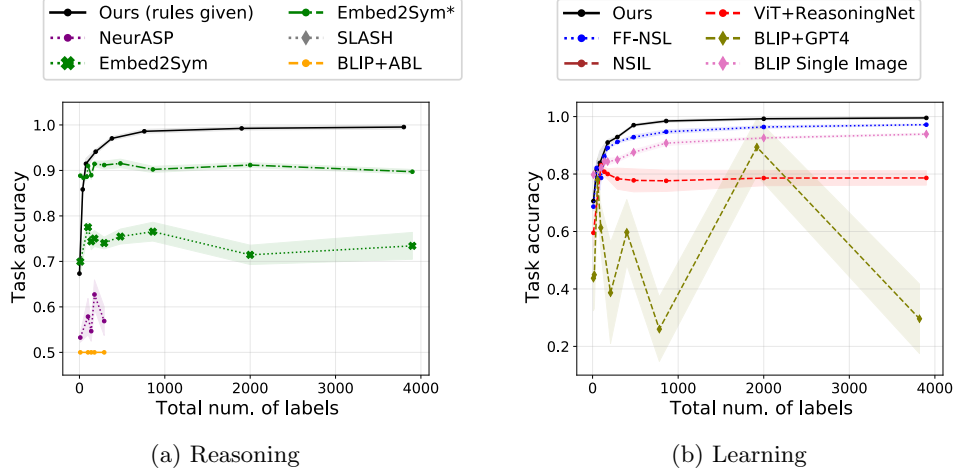


Figure 6.15: Plant Hitting Sets task accuracy.

```

0 {hs(V1,V2) } 1 :- elt(V2), hs_index(V1).
hit(V1) :- hs(V3,V2), ss_element(V1,V2), hs_index(V3), elt(V2), ss(V1).
:- ss_element(V1,V2), not hit(V1), elt(V2), ss(V1).
:- hs(V3,V1), hs(V3,V2), V1 != V2, hs_index(V3), elt(V2), elt(V1).
:- hs(V1,V2), healthy(V2), hs_index(V1), elt(V2).

```

Figure 6.16: Plant Hitting Sets learned hypothesis.

", to denote if a set X in the collection has been hit. In the body of the rule(s), consider using: (1) the "hs" predicate, (2) the "!=" relation to denote that two elements are not equal, (3) the "ss_el" predicate, (4) the "hit" predicate, and (5) a "healthy" predicate to constrain the hitting set to only include non-healthy elements. You may use all necessary ASP constructs such as variables, negation, constraints and choice rules. In ASP, negation of an atom p is expressed as "not p ". Negation of a predicate $q(x)$ is expressed as "not $q(x)$ ". For choice rules, use the form " $a \{ p(1); p(2); p(3) \} b$." which is a disjunction and denotes "choose a minimum of a , and a maximum of b of elements from the set $\{ p(1), p(2), p(3) \}$ ". Don't use any weak constraints, optimisation statements, square brackets, or # statements. Include a constraint to ensure that the same elements occur at the same index in the hitting set. Also ensure that each set in the given collection is hit. Make sure all variables introduced in the head of a rule exist in the body, and that all variables are grounded by the background knowledge or the set elements. Try to generate an ASP program that covers as many of the training examples as possible, whilst minimising the length of the program. Return your response as ##### <ASP program> #####.

Listing 6.4: BLIP+GPT4 prompt, Plant Hitting Sets task.

The results are shown in Figure 6.15 for both reasoning and learning. With > 10 labels, NeSyGPT is able to match or out-perform all the baseline methods. BLIP is fine-tuned to an accuracy of 0.8826 with 380 image labels, and 0.9871 with 3800 labels. The correct hypothesis is learned using the BLIP model fine-tuned with 380 image labels, when 100 examples are given to the symbolic learner (see Figure 6.16). Note the additional constraint is also learned (the last rule), which states that no healthy crops should occur in the hitting set. The other rules follow the same format as presented in Chapter 4.

In terms of the baselines, when the hypothesis is given, NeurASP and BLIP+ABL suffer similar timeout issues to those encountered in the Follow Suit Winner domain, and the implementation of SLASH does not support variable length sequences in the examples, so results cannot be obtained. Both variants of Embed2Sym are fairly constant, regardless of the number of labels. Inspecting the embedding space of the perception model, it appears the pre-trained vision transformer model is able to return a reasonably good image clustering without any fine-tuning, which explains why the results in both variants are fairly constant. Figure 6.17 shows the embedding space plots before training, and after training with both variants. In the before

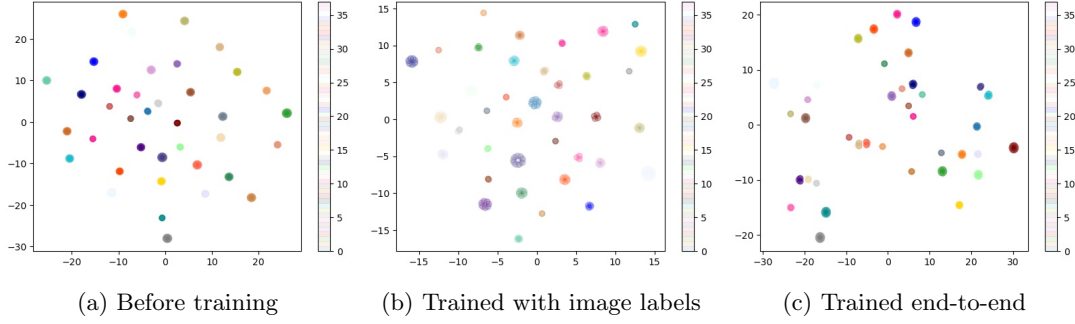


Figure 6.17: Embed2Sym embedding spaces — Plant Hitting Sets.

training case, we generate the TSNE plot with the 100-data point dataset that the cluster labelling stage observes, whereas after training, we use the images in the full dataset of 3800+100 data points. In the end-to-end case, the training data points in the embedding space are closer together. This makes it more difficult for K-means to identify general clusters in the data, to enable robust test-time performance. This explains why the performance of the Embed2Sym* variant is better than the end-to-end variant in Figure 6.15a. However, despite Embed2Sym* learning a reasonable clustering of the data, labelling the clusters accurately is very difficult, since there are many possible choices for the values of the concepts that satisfy the downstream label. This is why the performance of Embed2Sym* is lower than NeSyGPT in Figure 6.15a.

When learning the hypothesis, FF-NSL performs similarly to NeSyGPT, as does BLIP Single Image. The ViT+ReasoningNet is unable to solve the task, although when the reasoning network (a Recurrent Transformer) is given 5000 downstream examples ($50\times$ more than shown in Figure 6.15b), the task accuracy approaches 90%. The results for BLIP+GPT4 are very variable, and somewhat random. We note that this method requires extensive prompt tuning, although is able to learn the correct rules on some repeats. We had to limit the number of downstream examples to a maximum of 20, as otherwise the token limit in the prompt was exceeded. For this baseline, it is interesting to analyse the ASP programs that GPT4 generates. Listing 6.5 shows an example where the correct hypothesis is learned. This can be compared to the program learned by ILASP, presented in Figure 6.16. GPT4 generates the constraint that the hitting set must only contain healthy elements in the choice rule, which is a more compact representation than what ILASP learns (due to the representation given in the mode declarations). Although, GPT4 repeats the healthy constraint in the third and fourth rules, which is redundant. GPT4 also generates a constraint which states that the hitting set must not contain duplicate elements at the different indices, whereas ILASP learns the constraint that ensures different elements are not present at the same index. The third and fourth rules generated by GPT4 are similar to what ILASP learns, and a constraint that ensures each set in the collection is hit. Note that a definition of what it means for a set to be hit is included in the prompt, which was necessary in order to learn these rules.

```
% Choice rule for hitting set elements
1 { hs(I,E) : hs_index(I), elt(E), not healthy(E) } 2.

% Constraint for same elements to occur at same index
:- hs(I,E1), hs(J,E2), I != J, E1 = E2.

% Each set should be hit by at least one element from the hitting set
:- ss_el(S,E), not hit(S), elt(E), not healthy(E).

% A set is hit if it contains an element also in the hitting set
hit(S) :- hs(I,E), ss_el(S,E), elt(E), not healthy(E).
```

	Num Im.	Task Acc. (250 Task Ex.)	Task Acc. (500 Task Ex.)
α ILP	70k	0.9505 (0.0024)	0.9752 (0.0081)
	9k	0.4992 (0.0116)	0.4808 (0.0160)
NeSyGPT	70k	0.9853 (0.0012)	0.9853 (0.0012)
	9k	0.8629 (0.0090)	0.8691 (0.0085)

Table 6.1: CLEVR-Hans results.

Listing 6.5: Example GPT4 output for Plant Hitting Sets.

6.3.5 CLEVR-Hans

To demonstrate NeSyGPT can solve tasks that require detecting multiple objects within an image, we present results on the CLEVR-Hans3 dataset (Stammer et al., 2021). The input is a single image containing up to 10 objects, and the downstream label is one of three classes, which groups images with certain objects and properties together. An example data point is shown in Figure 6.18.

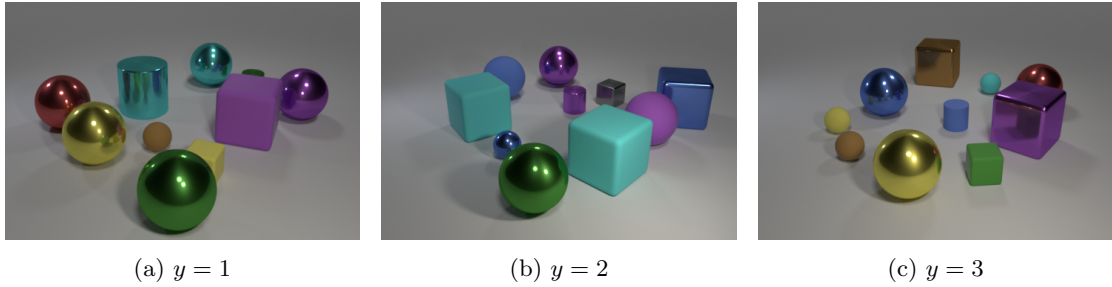


Figure 6.18: CLEVR-Hans3 example data points. The meta-data M is empty. Class 1 images contain a large cube and a large cylinder. Class 2 images contain a small metal cube, and a small sphere. Class 3 images contain a large blue sphere, and a small yellow sphere.

To solve this task, BLIP must detect the properties of the various objects, namely, their *size*, *material*, *color*, and *shape*. The learned hypothesis then classifies a set of objects and their properties as one of the three classes. The search space for the symbolic learner contains all possible properties and their values. BLIP is fine-tuned using the question “*what does the image contain?*” and the answer contains the list of object properties, e.g., “*The image contains a large gray cube made of metal, a large yellow cylinder made of rubber, ...*”. Image and task accuracies are reported over the CLEVR-Hans test set, where we count an image prediction as correct if all ground truth objects and their properties are correctly identified. For training, we fine-tune BLIP first using the 70k-example CLEVR dataset, as performed in (Shindo et al., 2023), and then, using the 9000-example CLEVR-Hans training set. For rule learning, we use 500 and 250 task examples. In all cases, we run 5 repeats with different randomly generated seeds. For α ILP, we use the 5 seeds specified in their implementation (Shindo et al., 2023).

```

label(0) :- size(V0,large), shape(V0,cube), size(V1,large), shape(V1,cylinder).
label(1) :- size(V0,small), material(V0,metal), shape(V0,cube), size(V1,small), shape(V1,sphere).
label(2) :- size(V0,large), color(V0,blue), shape(V0,sphere), size(V1,small), color(V1,yellow),
            shape(V1,sphere).

```

Figure 6.19: CLEVR-Hans learned hypothesis.

The results are shown in Table 6.1. NeSyGPT outperforms α ILP in all cases, achieving state-of-the-art performance. Note that NeSyGPT only requires 50% of the labelled task examples to do so. BLIP is fine-tuned to an image accuracy of 0.410 and 0.934, and the average hypothesis accuracy with 250 task examples is 0.831 and 1.0, for the 9000 and 70k cases respectively. With 500 task examples, the average hypothesis accuracy increases to 0.833 in the 9000 case. The correct hypothesis learned in the 70k case is shown in Figure 6.19, where V0 and V1 are object indices.

6.3.6 Reducing the Manual Engineering using a LLM

At the start of this chapter, we discussed a set of desiderata for an ideal neuro-symbolic system. We believe that an ideal system should balance control against autonomy, and enable the user to tune this balance for their particular use case. Our experiments so far have demonstrated that NeSyGPT outperforms many baselines in terms of accuracy, when learning in low-label settings. Also, NeSyGPT enables a greater amount of control over the learning task, since the perception component is fine-tuned with labelled values for the concepts, and the symbolic learner learns interpretable hypotheses that can be modified, if necessary. Our motivation was to solve the drawbacks of our previous FF-NSL approach, namely, (1) reducing the amount of labelled data required for training the perception component, and (2) reducing the amount of manual engineering. As discussed, our results demonstrate we have addressed (1), and (2) is partially addressed through the use of BLIP, since BLIP is a general architecture capable of solving many different perception tasks. The user no longer has to engineer a bespoke architecture for a given task (e.g., a CNN). However, the user still has to engineer the QA Generator, and the symbolic Example Generator. Therefore, we now attempt to automate the engineering of these components, by evaluating GPT4 (OpenAI et al., 2023) when tasked with generating questions and answers for fine-tuning BLIP, and when generating the training examples for the symbolic learner, given BLIP predictions. This enables the user to increase the autonomy of the system, if desired. Note that both the QA Generator and the symbolic Example Generator output interpretable representations, so it is possible for the user to edit these after GPT4 generation, to retain control over the learning process. In both cases, we prompt GPT4 with instructions and an example. Let us now present the prompts used in each task.

QA Generator. We use a base prompt, followed by an instruction for the given task. In this case, we require the user to input a natural language description of what the image represents, the space of possible concept values, and a description of the downstream task. Note that we do not specify the specific hypothesis of the task in the description. The base prompt used across all tasks is shown in Listing 6.6, and the specific instructions used for the Two-Digit Arithmetic, Follow Suit, and Plant Hitting Sets tasks are shown in Listings 6.7, 6.8, and 6.9 respectively.

You are required to generate a set of questions and answers to extract symbolic features from an image. You are given a description of the type of object the image represents, a description of the downstream task to be performed based on extracted features, and a list of ground-truth labels which denote the possible object instances. Return your response as a JSON list of questions and answers for each label. Try to use the exact same question(s) for all object instances, and ensure the answers are unique across each instance such that a classification can be performed. Under no circumstances should you return a response where two different labels have the same answers to the same question(s). Simple questions that distinguish between all instances are preferred over detailed questions on specific features. Use as few questions as possible, and eliminate redundant questions. Here is an example:

Description: The image represents a vehicle.

Downstream task description: The task is to learn rules based on the type of vehicle.

Labels: [car, ship, bicycle].

Output:

```

“‘json
{
  "car": [
    {"question": "Does the vehicle have wheels?", "answer": "yes"},
    {"question": "Does the vehicle drive on a road?", "answer": "yes"},
    {"question": "Does the vehicle carry passengers?", "answer": "yes"}
  ],
  "ship": [
    {"question": "Does the vehicle have wheels?", "answer": "no"},
    {"question": "Does the vehicle drive on a road?", "answer": "no"},
    {"question": "Does the vehicle carry passengers?", "answer": "yes"}
  ],
  "bicycle": [
    {"question": "Does the vehicle have wheels?", "answer": "yes"},
    {"question": "Does the vehicle drive on a road?", "answer": "yes"},
    {"question": "Does the vehicle carry passengers?", "answer": "no"}
  ]
}
“‘

```

Listing 6.6: Base prompt used across all tasks for generating QA for fine-tuning BLIP.

```

Generate a set of questions and answers as follows.
Description: The image represents a handwritten digit 0-9.
Downstream task description: Perform arithmetic over digits.
Labels: [0,1,2,3,4,5,6,7,8,9].

Output:

```

Listing 6.7: Instruction used for MNIST Arithmetic QA generation.

```

Generate a set of questions and answers as follows.
Description: The image represents a playing card from a standard deck of 52 cards.
Downstream task description: Learn the rules of a card game.
Labels: ["two of hearts", "two of diamonds", "two of spades", "two of clubs", ..., "ace of clubs"].

Do not truncate the response.

Output:

```

Listing 6.8: Instruction used for Follow Suit QA generation.

```

Generate a set of questions and answers as follows.
Description: The image represents a plant leaf. The plant leaf is from a certain species and may be diseased or healthy.
Downstream task description: Learn rules over groups of plant species and diseases.
Labels: ["apple scab", "apple black rot", ..., "tomato healthy"].

Output:

```

Listing 6.9: Instruction used for Plant Hitting Sets QA generation.

Example Generator. Again, we use a base prompt followed by an instruction specific to each task. We require the user to input a description of the downstream task, with details of the space of the concept values, and the range of output values. The ASP background knowledge and mode declarations are also required, and we also input the full question-answer JSON that is either manually specified, or LLM generated (see above). The base prompt is shown in Listing 6.10, and the instructions used for the Two-Digit Arithmetic, Follow Suit, and Plant Hitting Sets tasks are shown in Listings 6.11, 6.12, and 6.13 respectively. To avoid duplication, we omit the ASP background knowledge and mode declarations, as these are presented in Appendix A.

```

The task is to generate a python function called "create_wcdpi" that takes as input a list of image features in the form of questions and answers, together with a label and a unique identifier, and returns a training example for a symbolic learning system based on Answer Set Programming (ASP). The symbolic learning system

```

learns a set of logical rules in the form of an ASP program to solve a downstream task. A training example can be positive or negative, and it consists of the unique identifier, an integer weight, a set of inclusions, exclusions, and a set of contextual facts. The inclusion set can represent the label that should be output by the learned rules, given the contextual facts. The exclusion set can represent the labels that should *not* be output given the contextual facts. For binary classification tasks, the label can be represented with a positive (1) or negative (0) example, and the inclusion and exclusion sets can remain empty. Note the atoms within the inclusion and exclusion sets, as well as the atoms in the contextual facts are specified in the language of ASP. To generate a symbolic training example, the WCDPI python class can be used. This is instantiated as follows: `new_example = WCDPI(f"id{example_identifier}", positive=True, weight=10, inclusion=[], exclusion=[], context=[])`. The "example_id" attribute is the unique identifier, the boolean "positive" attribute indicates whether the example is positive or negative, the weight is the integer weight and should be set to 10, and the remaining attributes correspond to the inclusion, exclusion, and context sets respectively. To help generate the correct python function, you are given a description of the downstream task, a complete set of possible questions and answers for each type of image, as well as an ASP background knowledge specification related to the downstream task. Finally, you are also given a set of mode declarations, which the symbolic learner uses to construct a search space for possible rules. The mode declarations define what should be in the head and the body of the learned rules, using #modeh, and #modeb statements respectively. The background knowledge and mode declarations give clues as to the format of the atoms in the inclusion and exclusion sets, as well as the contextual facts.

Here are some examples:

Example 1:

Description: The downstream task is to learn a set of rules define the validity of 4x4 Sudoku grids. Image features are given related to the digits apparent in each cell. The digit values are in the range 1-4, and the output is a binary classification. Note that it is important to associate the value of each digit to a particular cell on the Sudoku grid.

Complete set of questions and answers for each type of image: {"1": [{"question": "what is the digit value?", "answer": "1"}], "2": [{"question": "what is the digit value?", "answer": "2"}], "3": [{"question": "what is the digit value?", "answer": "3"}], "4": [{"question": "what is the digit value?", "answer": "4"}]}

ASP background knowledge for the downstream task:

```
value("1, 1", X) :- value(1, 1, X).
value("1, 2", X) :- value(1, 2, X).
value("1, 3", X) :- value(1, 3, X).
value("1, 4", X) :- value(1, 4, X).
value("2, 1", X) :- value(2, 1, X).
value("2, 2", X) :- value(2, 2, X).
value("2, 3", X) :- value(2, 3, X).
value("2, 4", X) :- value(2, 4, X).
... etc.
```

% Columns

```
col("1, 1", 1).
col("1, 2", 2).
col("1, 3", 3).
col("1, 4", 4).
col("2, 1", 1).
col("2, 2", 2).
col("2, 3", 3).
col("2, 4", 4).
... etc.
```

% Rows

```
row("1, 1", 1).
row("1, 2", 1).
row("1, 3", 1).
row("1, 4", 1).
row("2, 1", 2).
row("2, 2", 2).
row("2, 3", 2).
row("2, 4", 2).
... etc.
```

% Blocks

```
block("1, 1", 1).
block("1, 2", 1).
```

```

block("2, 1", 1).
block("2, 2", 1).
... etc.

Mode declarations for the symbolic learner:
#modeh(invalid).
#modeb(value(var(cell), var(num))).
#modeb(row(var(cell), var(row))).
#modeb(not row(var(cell), var(row))).
#modeb(col(var(cell), var(col))).
#modeb(not col(var(cell), var(col))).
#modeb(block(var(cell), var(block))).
#modeb(not block(var(cell), var(block))).
#modeb(neq(var(cell), var(cell))).
neq(X, Y) :- cell(X), cell(Y), X != Y.

#maxv(4).
num(1..4).
row(1..4).
col(1..4).
block(1..4).
cell(C) :- value(C, _).

#bias("penalty(1, head).").
#bias("penalty(1, body(X)) :- in_body(X).").

Python function:

'''python
def create_wcdpi(example_identfier: str, image_features: list, label: str):
    # Decide whether to create a positive or negative example
    if label == '1':
        positive = True
    else:
        positive = False

    # Create contextual facts
    # Assume image features are passed in order of increasing row and column
    ctx = []
    GRID_SIZE = 4
    current_idx = 0
    for row in range(1, GRID_SIZE+1):
        for col in range(1, GRID_SIZE+1):
            ctx.append(f'value({row},{col},{image_features[current_idx]["what is the digit value?"]}).')
            current_idx += 1

    # Create WCDPI
    new_example = WCDPI(ex_id=f"id{example_identfier}", positive=positive, weight=10, inclusion=[], exclusion=
        [], context=ctx)

    return new_example
'''

Example 2:
Description: The downstream task is to learn a set of rules that define the features of various animals. Image
features are given related to a single animal (one image). Only one animal is present in each image.
Complete set of questions and answers for each type of image: {"fish": [{"question": "does it have hair?", "
answer": "no"}, {"question": "does it have fins?", "answer": "yes"}, {"question": "does it lay eggs?", "
answer": "yes"}, {"question": "does it have feathers?", "answer": "no"}, {"question": "does it have legs
?", "answer": "no"}], "bird": [{"question": "does it have hair?", "answer": "no"}, {"question": "does it
have fins?", "answer": "no"}, {"question": "does it lay eggs?", "answer": "yes"}, {"question": "does it
have feathers?", "answer": "yes"}, {"question": "does it have legs?", "answer": "yes"}], "mammal": [{"
question": "does it have hair?", "answer": "yes"}, {"question": "does it have fins?", "answer": "no"}, {"
question": "does it lay eggs?", "answer": "no"}, {"question": "does it have feathers?", "answer": "no"},
{"question": "does it have legs?", "answer": "yes"}]}

ASP background knowledge for the downstream task:
:- class(X), class(Y), X < Y.

Mode declarations for the symbolic learner:
#maxv(1).

```

```
#modeh(class(const(class))).
#modeb(1, hair(const(hair))).
#modeb(1, feathers(const(feathers))).
#modeb(1, eggs(const(eggs))).
#modeb(1, fins(const(fins))).
#modeb(1, legs(const(legs))).

Python function:

'''python
def create_wcdpi(example_identifier: str, image_features: list, label: str):
    # Inclusion set
    inclusion_set = [f'label({label})']

    # Exclusion set - all other classes
    classes = ['fish', 'bird', 'mammal']
    exclusion_set = [f'label({l})' for l in classes if l != label]

    # Create contextual facts
    # Single image so extract first image features
    first_image = image_features[0]
    ctx = [f'hair({first_image["does it have hair?"]})', f'fins({first_image["does it have fins?"]})',
           f'feathers({first_image["does it have feathers?"]})', f'eggs({first_image["does it lay eggs?"]})',
           f'legs({first_image["does it have legs?"]})']

    # Create WCDPI
    new_example = WCDPI(ex_id=f'id{example_identifier}', positive=True, weight=10, inclusion=inclusion_set,
                        exclusion=exclusion_set, context=ctx)

    return new_example
'''
```

Listing 6.10: Base prompt used across all tasks for generating symbolic training examples.

```
Goal:
Generate a Python function for the following task:
Description: The downstream task is to learn a set of rules that perform arithmetic over two numeric digits.
            Image features are given related to the two digits, and the digit values are in the range 0-9. The output
            is in the range 0-18.

Complete set of questions and answers for each type of image:
{'0': [{'question': 'which digit is this?', 'answer': '0'}],
 ...,
'9': [{'question': 'which digit is this?', 'answer': '9'}]}

<<Insert ASP background knowledge and mode declarations>>

Python function:
```

Listing 6.11: Instruction used for Two-Digit Arithmetic symbolic example generation.

```
Goal:
Generate a Python function for the following task:
Description: The downstream task is to learn a set of rules of a playing card game with 4 players. Image
            features are given related to 4 playing card images, which correspond to a card played by each player of
            the game. The cards take values from a standard deck of 52 playing cards, and the output indicates which
            player wins the game.

Complete set of questions and answers for each type of image: {'2h': [{'question': 'which color are the symbols
?', 'answer': 'red'}, {'question': 'which symbol is on the card?', 'answer': 'heart'}, {'question': 'which
suit is the playing card?', 'answer': 'hearts'}, {'question': 'which rank is the playing card?', 'answer
': '2'}, {'question': 'is it a number card or face card?', 'answer': 'number'}],
 ...,
'ac': [{'question': 'which color are the symbols?', 'answer': 'black'}, {'question': 'which symbol is on the
card?', 'answer': 'club'}, {'question': 'which suit is the playing card?', 'answer': 'clubs'}, {'question
': 'which rank is the playing card?', 'answer': 'ace'}, {'question': 'is it a number card or face card?',
'answer': 'face'}]}

<< Insert ASP background knowledge and mode declarations>>
```

	Two-Digit Arithmetic 100+100 labels		Follow Suit Winner 4 52+100 labels		Follow Suit Winner 10 52+100 labels		Plant HS 760+100 labels
	Sum	E9P	Standard	ACA	Standard	ACA	
Manual QA + Manual Sym. Ex.	0.8492 (0.01)	0.9004 (0.00)	0.9958 (0.00)	0.9994 (0.00)	0.9968 (0.00)	1.0000 (0.00)	0.9846 (0.00)
LLM QA + Manual Sym. Ex.	0.6954 (0.16)	0.7302 (0.17)	0.9928 (0.00)	0.9984 (0.00)	0.9892 (0.01)	0.9974 (0.00)	0.9734 (0.01)
Manual QA + LLM Sym. Ex.	0.5158 (0.21)	0.5486 (0.22)	0.2938 (0.20)	0.2938 (0.20)	0.2936 (0.18)	0.2936 (0.18)	0.7874 (0.20)
LLM QA + LLM Sym. Ex.	0.1634 (0.16)	0.1746 (0.17)	0.0500 (0.05)	0.0500 (0.05)	0.0908 (0.09)	0.0912 (0.09)	0.1974 (0.20)
Manual QA + LLM Sym. Ex.*	0.8492 (0.01)	0.9004 (0.00)	0.5986 (0.24)	0.5994 (0.24)	0.9968 (0.00)	1.0000 (0.00)	-
LLM QA + LLM Sym. Ex.*	0.6832 (0.17)	0.7198 (0.18)	0.8472 (0.15)	0.8486 (0.15)	0.7898 (0.20)	0.7976 (0.20)	-

Table 6.2: Task accuracy with manual vs LLM engineering. The number of labels is decomposed as “num image labels + num task labels”. * denotes manual fixes to simple errors.

Python function:

Listing 6.12: Instruction used for Follow Suit symbolic example generation.

```
Goal:
Generate a Python function for the following task:
Description: The downstream task is to learn a set of rules over a collection of sets containing images related
to plant disease. There are various plant diseases, and they are denoted by integers 1-38 inclusive. It is
important to associate each image with a particular set identifier. The output is a binary classification
. Assume a global variable "ss_element_assignments" is available, that is assigned a list that contains
subset assignments for each image in an example. This list can be indexed by the example identifier to
return a list of subset identifiers, one for each image. Also, assume a function "get_image_class" is
available, that takes as input the key "{species}_{disease}", and returns the associated integer in the
range 1-38.

Complete set of questions and answers for each type of image: {'1': [{'question': 'which plant species is this
? ', 'answer': 'apple'}, {'question': 'which disease is this?', 'answer': 'scab'}],
...,
'38': [{'question': 'which plant species is this?', 'answer': 'tomato'}, {'question': 'which disease is this?',
'answer': 'healthy'}]}}

<<Insert ASP background knowledge and mode declarations>>

Python function:
```

Listing 6.13: Instruction used for Plant Hitting Sets symbolic example generation.

In order to evaluate the responses from the LLM, we perform a variety of experiments with all combinations of manual and LLM engineered components, and report the mean task accuracy, and standard error over 5 repeats. We use 5 different random seeds, and pass these to the LLM accordingly. All experiments are performed with the OpenAI API, and queries were made between 19th-30th December 2023. For the QA generation prompts, we use the “gpt-4” model with 1000 max tokens for Two-Digit Arithmetic, and the “gpt-4-1106-preview” model with 4096 max tokens for Follow Suit Winner and Plant Hitting Sets. For the symbolic example prompts, we use the “gpt-4” model with 750 max tokens in all tasks. Note on the Follow Suit Winner task, we give only 50% of the labels in the QA JSON, as otherwise the max token limit is exceeded.

The results are shown in Table 6.2. Manually engineering everything achieves the best performance, although the LLM shows promise in generating questions and answers for fine-tuning BLIP in the Follow Suit Winner, and Plant Hitting Sets tasks (see *LLM QA + Manual Sym.Ex*). In the Two-Digit Arithmetic tasks, it is difficult to decompose the MNIST images into a set of questions. On four repeats, GPT4 returns the expected response of directly querying the value of the digit, whereas on one repeat it returns a set of questions based on the digit being even, and greater than 5. When GPT4 directly generates the function to create symbolic examples, the accuracy is poor, although we show with simple syntactic fixes, the accuracy is able to approach that of manual engineering in the Two-Digit Arithmetic and the 10-player Follow Suit Winner tasks. Simple errors include using a `solution` predicate in the inclusion set

<pre>def generate_example(example_id: int, blip_out: list[dict[str, str]], label: str, meta_data: list None = None) -> WCDPI: # Inclusion set - solution consists of label inclusion_set = [f'solution(_,_,{label})'] # Create contextual facts from the image features # Image features consist of two digit images ctx = [f'digit(1,{blip_out[0]["which digit is this?"]})', f'digit(2,{blip_out[1]["which digit is this?"]})',] # Create WCDPI return WCDPI(ex_id=f'id{example_id}', positive=True, weight=10, inclusion=inclusion_set, exclusion=[], context=ctx)</pre>	<pre>def generate_example(example_id: int, blip_out: list[dict[str, str]], label: str, meta_data: list None = None) -> WCDPI: # Inclusion set - solution consists of label inclusion_set = [f'label({label})'] # Create contextual facts from the image features # Image features consist of two digit images ctx = [f'digit(1,{blip_out[0]["which digit is this?"]})', f'digit(2,{blip_out[1]["which digit is this?"]})',] # Create WCDPI return WCDPI(ex_id=f'id{example_id}', positive=True, weight=10, inclusion=inclusion_set, exclusion=[], context=ctx)</pre>
--	---

(a) GPT-4 generated mistake

(b) Manual correction

Figure 6.20: Example mistake correction for the GPT-4 implementation of the Example Generator in the Two-Digit Arithmetic task. In this example, the mistake is using a `solution` predicate in the inclusion set, instead of the `label` predicate.

instead of the `label` predicate in the Two-Digit Arithmetic tasks (see Figure 6.20), and using the rank values “jack”, “queen”, “king”, and “ace”, instead of single letters, as expected by the ASP background knowledge in the Follow Suit Winner tasks. In the Plant Hitting Sets case, the responses contained significant errors that couldn’t easily be fixed manually. Requiring the user to apply simple fixes is a significantly reduced level of engineering compared to specifying the full components. In all but the Plant Hitting Sets domain, when using the LLM to generate both components, and assuming user fixes, the task accuracy achieved is reasonable. As LLMs are an active research area, we expect these results to improve as LLMs improve.

6.3.7 Zero-Shot Evaluation

We now consider whether more recent vision-language foundation models such as BLIP2 (Li et al., 2023a), and GPT4 Vision (GPT4V) (OpenAI, 2023) can be used out-of-the-box within the NeSyGPT architecture. These models do not currently support fine-tuning with VQA, but they do support VQA inference, and they are more advanced than the original BLIP approach used in our main results (BLIP1). To investigate this, we therefore generate image predictions in a zero-shot fashion via natural language questions (prompts in the case of GPT4 Vision), and then use these image predictions to perform rule learning in our downstream tasks. When performing rule learning, we present the mean task accuracy achieved over 5 repeats.

With both BLIP approaches, we consider asking the models to generate the image predictions directly, and on the Follow Suit and Plant Hitting Sets images, via a conditional questioning approach (denoted CQA). This is useful when the concepts can easily be decomposed into various questions. For example, in the playing cards domain, one can initially ask the question “which colour are the symbols?”, before asking “which symbol is on the card?”. Candidate answers can then be restricted accordingly, e.g., to force the model to return “hearts” or “diamonds” for the symbol question, if the colour is predicted as “red”. We do not use this technique for the MNIST images, as these are challenging to decompose.

We also evaluate the “generate” and “rank” inference modes in BLIP1, where the “generate” approach is the same as the one presented in our main results, with the string distance measure included in the case of image classification. The rank approach requires giving possible candidate

	MNIST	Playing Cards		Plant Disease
		Standard	ACA	
BLIP1 Generate	0.27	0.074	0.0096	0.0008
BLIP1 Rank	0.2262	0.0971	0.0567	0.035
BLIP1 Rank CQA	-	0.6962	0.225	0.0326
BLIP2 ITM	0.6521	0.7846	0.3096	0.1214
BLIP2 ITM CQA	-	0.6385	0.301	0.1624
GPT4 Vision*	0.5178	0.9048	0.8625	0.04

Table 6.3: Foundation model zero-shot image accuracy.

	Arithmetic		Follow Suit Winner 4		Follow Suit Winner 10		Plant HS
	Sum	E9P	Standard	ACA	Standard	ACA	
BLIP1 Generate	0.089	0.1484	0.4198	0.3038	0.174	0.1226	0.7688
BLIP1 Rank	0.0742	0.1018	0.301	0.2806	0.1784	0.1716	0.7624
BLIP1 Rank CQA	-	-	0.4278	0.2914	0.271	0.0808	0.7176
BLIP2 ITM	0.4276	0.5458	0.5074	0.3734	0.3168	0.1854	0.7252
BLIP2 ITM CQA	-	-	0.4706	0.2858	0.2688	0.115	0.8102
GPT4 Vision*	0.2602	0.2982	0.4502	0.4886	0.2026	0.235	-

Table 6.4: Foundation model task accuracy with zero-shot image predictions.

answers to BLIP, which then returns the answer that minimises the language modelling loss between each candidate and the generated output. We didn’t use this for NeSyGPT in the main results, because after fine-tuning, we obtained better results using the generate mode. For BLIP2, we evaluate the Flan-t5 XXL instruct, feature extractor, and Image-Text-Matching (ITM) modes, and select the ITM mode as this results in the best performance. For GPT4V, we use the OpenAI API with the model “gpt-4-vision-preview”, and set the max tokens parameter to 300. All responses were obtained when accessing the API between 19th-30th December 2023. Finally, in all tasks, we use 100 downstream examples.

The image and downstream task accuracy results are shown in Tables 6.3 and 6.4 respectively. As you can see, more recent models such as BLIP2 and GPT4V outperform BLIP1, but still fall significantly below the levels of accuracy achieved in the main results, where a small amount of fine-tuning was used with labelled data. Also, the correct hypotheses are only learned on the Two-Digit Arithmetic tasks when BLIP2 and GPT4V are used zero-shot, but again, the task accuracy is lower than that presented in our main results, when the hypotheses are learned.

The * indicates that GPT4V was unable to return a classification for all of the images in the datasets. We often observe error responses from the OpenAI API, including file extension errors, and errors regarding violation of the profanity filter. The error rates are 38.83% (MNIST), 8.08% (Playing Cards Standard), 4.23% (Playing Cards ACA), and 92.16% (Plant Disease). Note that on the Plant Disease dataset, most of the errors are because GPT4V refused to make a classification, citing a lack of information, rather than an error from the API. The accuracy results presented in Table 6.3 are measured over the full test set, and therefore include erroneous data points. In order to learn the hypotheses with the symbolic learner, examples without a complete set of image predictions are removed from the symbolic learning task. On the Plant Hitting Sets task with GPT4V, there are therefore no examples, and no rules can be learned. If a test example has missing image predictions, it is assumed incorrect.

The best results are on the Follow Suit tasks, although even with an image accuracy of 0.9048 from GPT4V on the standard deck, it is insufficient to learn the correct hypothesis. With more downstream task examples, it may be possible to do so, and improve the task accuracy. These results demonstrate that the fine-tuning approach of NeSyGPT is currently required to achieve

strong performance. In future, should BLIP2 or GPT4V support VQA fine-tuning, it may be possible to improve upon the NeSyGPT results presented in this chapter.

6.4 Summary

We have introduced a new architecture called NeSyGPT, that integrates a vision-language foundation model with symbolic learning and reasoning in ASP. We have demonstrated NeSyGPT achieves state-of-the-art performance in four learning tasks, outperforming many neuro-symbolic and fully neural baselines. Furthermore, NeSyGPT effectively solves the drawbacks of our previous approaches. NeSyGPT requires fewer data points than FF-NSL for training the perception component, and can scale to tasks with a significantly larger amount of choice for the values of the concepts, when compared to NSIL. NeSyGPT requires a reduced amount of manual engineering when compared to both approaches, since BLIP is a more general perception architecture capable of solving a wider range of tasks (e.g., object detection in the CLEVR-Hans task). We have also demonstrated that the amount of manual engineering can be reduced even further, by using a LLM to generate the “QA Generator”, and the symbolic “Example Generator” components.

When considering our desiderata introduced at the start of this chapter, we have demonstrated NeSyGPT: (1) Can scale to complex tasks with a large amount of choice for the values of the primitive concepts, since it decouples the training of the perception and reasoning components. (2) Is highly performant, learns optimal solutions in all of our evaluation tasks, and can solve tasks that require more advanced perception, such as object detection. This is challenging to achieve in many existing neuro-symbolic integrations, as they rely upon training a CNN architecture. (3) Learns interpretable solutions, in terms of the symbolic hypothesis, which is defined in terms of concept values. Interpreting the predictions of BLIP remains an outstanding challenge, and there is recent work aiming to address this (Chen et al., 2023). (4) Enables fine-grained control over the learning process, since BLIP is fine-tuned directly with concept labels. This avoids the problem of “reasoning shortcuts” when training end-to-end (Marconato et al., 2024). (5) Requires a significantly reduced amount of labelled data for training, when compared to fully neural and even neuro-symbolic baselines. We have also demonstrated NeSyGPT can be configured to automate the generation of the QA Generator and Example Generator components using a LLM, which reduces the amount of manual engineering required compared to other neuro-symbolic approaches. The level of autonomy can be balanced against the desire for controlling the learning process, which is important for deployment in high-stakes applications. In the next chapter, we review the neuro-symbolic literature, and highlight how our contributions in this thesis stand against the existing literature.

Chapter 7

Related Work

In this chapter, we analyse the Neuro-Symbolic literature relevant to our work. The field is interconnected among many areas within the broader AI community. For example, deep learning, ILP, logic, knowledge representation, probability, and Statistical Relational Learning for Artificial Intelligence (StarAI) all contribute various ideas and techniques. We divide our analysis into two broad sections: (1) *Directly Related Work*, which is required to understand the contributions of this thesis in the context of the most similar literature, and (2) *Other Related Work*, which broadly analyses additional literature, useful for considering future work.

7.1 Directly Related Work

In this section, we discuss works that are most relevant to the contributions of this thesis, in the area of *Neuro-Symbolic Learning*, where symbolic knowledge is learned from raw data. We include works that are closely related in terms of the similarity of the method, but also works that are used as baselines for comparison. These approaches can be broadly divided into two categories. (1) Hybrid approaches, which are most similar to the methods presented in this thesis, that retain a clear separation between the neural and symbolic components, with the knowledge being learned symbolically, and (2) Differentiable approaches, that aim to learn symbolic knowledge in a form that can easily integrate with neural network training. Let us now discuss the main approaches within each category.

7.1.1 Hybrid Approaches

Meta_{Abd} (Dai and Muggleton, 2021) was one of the first approaches to attempt simultaneous neuro-symbolic learning. *Meta_{Abd}* extends the ABL framework (Dai et al., 2019) (see Section 7.2) by using abduction and induction to jointly train a neural network and learn symbolic knowledge from raw data. Specifically, both the knowledge and concept pseudo-labels for training the neural network are learned and abduced respectively, in one step. However, the symbolic learner used by *Meta_{Abd}* is restricted in the expressivity of knowledge that can be learned (Law, 2018). In contrast, our systems can learn ASP programs to represent common-sense knowledge involving defaults, exceptions, constraints and choice, all of which are outside the scope of *Meta_{Abd}*. Using ASP also enables our learned programs to have multiple answer sets (multiple models), whereas *Meta_{Abd}* is restricted to learning definite programs that accept only one model. Aside from the differences in the symbolic component, the neuro-symbolic integration presented by *Meta_{Abd}* does not appear to cater for the case where the incorrect knowledge is initially learned.

The search spaces used are highly restrictive, and in their evaluation, the correct knowledge is always learned immediately. When comparing *Meta_{Abd}* to our closest system, NSIL, we demonstrate superior performance in significantly larger search spaces (see Chapter 4). We have demonstrated that our approach of biasing the symbolic learner using corrective examples enables NSIL to converge to a more accurate hypothesis than the hypothesis learned initially, offering a significant advantage compared to *Meta_{Abd}*. Finally, as discussed in (Tsamoura et al., 2021), ABL is highly sensitive to the initialisation of the neural network parameters. As a result, the performance of *Meta_{Abd}* using different random seeds varies greatly. Meanwhile, we demonstrate NSIL has robust performance over 20 repeats with different, randomly generated seeds.

The NeuralFastLAS approach introduced in (Charalambous et al., 2023) is also a hybrid technique that retains separation between a neural network, and the FastLAS symbolic learner. NeuralFastLAS first computes the opt-sufficient subset of the hypothesis space using the standard FastLAS algorithm, and then trains the neural network using the semantic loss function (Xu et al., 2018). The neural network also learns a posterior probability distribution over the set of opt-sufficient rules, and an ASP solving stage computes the final solution, by optimising a joint objective based on the length of the hypothesis, and the scores of each rule as learned by the neural network. Whilst NeuralFastLAS is guaranteed to learn an optimal solution, it is limited to the FastLAS symbolic learner, and is therefore only able to solve observational predicate learning tasks that require learned concepts to be defined in the examples. Our NSIL approach on the other hand, treats the LAS symbolic learner as a black-box, and thus can utilise the expressivity of the ILASP symbolic learner to solve more complex tasks. Also, since NeuralFastLAS is based upon the semantic loss, it is unlikely to scale to tasks with a large amount of choice for the concept values. In Chapter 6, we have demonstrated that our decoupled NeSyGPT approach can solve such tasks with a greater accuracy than other approaches that leverage the semantic loss for neural network training.

A recent technique presented in (Li et al., 2023c) also trains both neural and symbolic components in a hybrid fashion. Network predictions enable Boolean cardinality constraints to be learned via a (non-differentiable) proximal point algorithm (Rockafellar, 1976). The neural network predictions are then revised to ensure constraint satisfiability, and the revised predictions are used to update the network parameters. To avoid trivial constraint learning, (Li et al., 2023c) employ a trust region approach, which enables the system to search for alternative constraints within a given region of uncertainty (Boyd and Vandenberghe, 2004). This effectively balances exploration vs exploitation, with similar aims as our corrective example approach in NSIL. Since Boolean cardinality constraints are learned, they can be *exactly* enforced during inference, using an off-the-shelf SMT solver. Whilst (Li et al., 2023c) are able to prove convergence, and demonstrate impressive experimental results, they are limited to learning Boolean cardinality constraints, which are expressed in propositional logic. Our approaches on the other hand, are able to learn first-order rules, expressed in ASP, which are significantly more expressive.

Another hybrid approach is based upon class expression learning (Demir and Ngomo, 2023). Class expression learning is similar to ILP, as it aims to find a class expression, that correctly separates positive and negative examples, consistent with a large knowledge base. The neuro-symbolic method presented in (Demir and Ngomo, 2023) learns heuristic functions via deep Q-learning, to guide the symbolic search procedure towards goal states. This has synergies with our corrective example method of NSIL. However, our approach guides the search via weighted (and automatically generated) examples for the symbolic learner, rather than heuristic functions. Also, class expression learning learns very specific rules that separate a given dataset, whereas

the ILP techniques used in our approaches learn general knowledge that can be applied to new datasets. Also, it is not clear whether class expression learning can handle noisy examples, and indeed the evaluation in (Demir and Ngomo, 2023) is only performed over noise-free symbolic datasets.

Integrating neural networks with symbolic learning systems is challenging, since the symbolic learner is not differentiable. Therefore, many approaches tackle the integration by pursuing a differentiable method of learning knowledge. The idea is to support seamless integration with neural networks and train the entire architecture end-to-end via gradient descent.

7.1.2 Learning Knowledge in a Differentiable Manner

One of the first attempts at differentiable rule learning was ∂ ILP (Evans and Grefenstette, 2018), which re-implements a top-down, generate and test ILP approach in an end-to-end differentiable manner. Top-down approaches *generate* clauses from a language definition referred to as a mode bias, before *testing* these clauses on positive and negative examples. Some top-down approaches use Boolean flags to indicate which clauses to turn on and off. This transforms the induction problem into a satisfiability problem: Choose an assignment to the Boolean flags such that the turned on clauses together with the background knowledge satisfy the positive examples and none of the negative examples (Corapi et al., 2010). ∂ ILP extends this approach, by contributing a differentiable architecture that leverages gradient descent to learn which clauses to turn on and off. Boolean flags are instead replaced with continuous weights in the real unit interval $[0, 1]$ that determines a probability distribution over possible clauses. ∂ ILP then interprets the satisfiability problem as binary classification for accepting or rejecting an example. A cross-entropy loss function compares the predicted label for an example (i.e., whether it is accepted or rejected by the learned rules) with a ground truth label. The calculated loss can then be differentiated with respect to the weights of generated clauses, to enable the weights to be learned. ∂ ILP can be applied to traditional symbolic ILP benchmark tasks, as well as neuro-symbolic tasks where they rely upon a pre-trained neural network to extract concepts from images. This is therefore closely related to our FF-NSL approach presented in Chapter 3. However, the framework has a number of disadvantages. First, as the authors state, the framework is highly memory intensive due to the number of clauses and predicates generated. As a result, the authors limit ∂ ILP to nullary, unary and binary predicates. This also restricts ∂ ILP when scaling to larger datasets and the authors state that ∂ ILP cannot be used on the Mutagenesis data set¹, which contains only 230 examples. In contrast, our approaches have no such restrictions and can be applied to larger datasets.

Given these drawbacks, (Shindo et al., 2021) extends ∂ ILP with various techniques. First, they introduce a clause refinement stage, which uses a beam search to find clauses for the symbolic search space that are likely to lead to accurate classification results. Second, they introduce an adaptive fact enumeration, that utilises the training examples, and the background knowledge, to calculate a smaller set of ground atoms than in ∂ ILP. These ground atoms define the size of the tensors used during the inference stage, and smaller tensors are desirable. This leads to a performance improvement compared to ∂ ILP, which uses the full Herbrand Base. Finally, they introduce a differentiable inference function which assigns multiple weights to each clause, instead of assigning weights to pairs of clauses. Their approach can efficiently approximate logic programs, which also provides a performance improvement compared to ∂ ILP. This approach can be applied to more complex problems that require reasoning over tree structures and sequences.

¹<https://relational.fit.cvut.cz/dataset/Mutagenesis>

However, they do not learn from raw data such as images.

Therefore, α ILP (Shindo et al., 2023) lifts the ILP approach in (Shindo et al., 2021) to the neuro-symbolic setting, and solves tasks requiring perception from visual scenes. (Shindo et al., 2023) uses a pre-trained neural network to extract concepts in the form of an object-centric representation, from raw images. The concepts are then converted into probabilistic facts, to enable a differentiable forward reasoning step to be performed using a set of weighted clauses. The clauses are generated from a beam search over the hypothesis space, and the weights of each clause are optimised using the loss function with respect to the final ground truth labels. The main limitation of α ILP is that their ILP approach is restricted to learning definite clauses (Shindo et al., 2021). In contrast, the symbolic learners used in this thesis can learn more expressive knowledge involving constraints, choice, and programs with multiple models, in the language of ASP. Also, as shown in Chapter 6, our NeSyGPT approach is able to outperform α ILP on the CLEVER-Hans task, with significantly fewer labelled data points.

The Logical Neural Network (LNN) framework (Riegel et al., 2020), is also capable of learning logical rules (Sen et al., 2022) in a differentiable fashion. In the original LNN framework, a neural network architecture is constructed to match the semantics of a given logic program. The output activations of each neuron are restricted to the range $[0, 1]$, and the semantics of each neuron correspond to a given logic gate. This is a hard constraint that provides a formal guarantee that the resulting network will conform to the semantics of the given logic program. In (Sen et al., 2022), the network is able to *learn* which logic gate to use for each neuron during training, so it can effectively learn which rules to apply. However, this approach requires hand-crafted rule templates, which are a highly restrictive method of specifying the symbolic search space. In contrast, the symbolic learners used in our work accept a set of mode declarations, which enables a much broader specification of the symbolic search space. Also, the LNN framework is limited in terms of the expressivity of knowledge that can be learned or represented. Only conjunction, disjunction, and negation can be used, and the LNN does not support more advanced constructs such as recursion, general constraints, choice rules, or programs with multiple models, as learned by our systems presented in this thesis.

The Embed2Sym framework (Aspis et al., 2022) is also directly related to our contributions, since it has been extended to support rule learning (Aspis et al., 2024). The original framework assumes a complete symbolic knowledge base is given, and first trains a fully neural architecture to solve the perception and reasoning task, using a trainable multi-layer perceptron in place of the symbolic knowledge. The key observation, is that under fully differentiable training, the perception network learns to cluster the raw inputs into distinct regions in the embedding space, where each cluster represents a concept value. The multi-layer perceptron is then discarded, and an ASP optimisation is employed to assign concept labels to the clusters, given the symbolic knowledge and the downstream task label for a set of training examples. Embed2Sym is most closely related to our contributions in Chapter 5, that also rely upon image clustering techniques for training the neural network. Whilst Embed2Sym achieves state-of-the-art performance in cases where the symbolic knowledge is given, it becomes challenging to train the perception network, and label the clusters, in tasks with a large number of possible concept values. We hypothesise this is due to a weakening training signal (see our analysis in Chapter 4). The extension to rule learning, called Embed2Rule (Aspis et al., 2024), adopts the BLIP vision-language foundation model to perform weak labelling of the learned clusters. In this architecture, BLIP is prompted (without fine-tuning) to make a concept prediction for each image, and the Hungarian matching algorithm is used to select a single concept label for all images within each cluster based on a majority vote. At this point, the concepts can be input to a symbolic learning

system to learn the rules of the downstream task. Whilst this achieves neuro-symbolic learning without human annotated concept labels, it is restricted to the general world knowledge available within BLIP, and is not able to support weak labelling over images that are not in the public domain. This limits its adoption in industry, and it is unclear how this approach would perform on private datasets. Also, since it relies upon clustering, it is not clear whether tasks such as object detection could be supported, unless the clusters were discarded and BLIP was used zero-shot. In contrast, our NeSyGPT approach that also uses BLIP, can support learning with private data (since we adopt fine-tuning), and can support object detection with the CLEVR-Hans dataset.

Another approach, called Deep Symbolic Learning (DSL) (Daniele et al., 2023), uses policy functions inspired by Reinforcement Learning, to select concept values for each raw input, based on the neural network confidence. DSL also applies policy functions to learn how to predict the target output, by weighting each possible function and learning the weights through end-to-end training. Note that DSL also learns how to predict the concepts through a similar technique. Whilst this method achieves good performance in the MNIST Arithmetic domains, the knowledge learned is not general, and essentially constitutes a look-up table, which maps combinations of concept values to the target value. In contrast, our approaches all learn general and expressive knowledge, which can be applied on various downstream tasks that have slightly different requirements (e.g., the Hitting Set problem with a greater number of subsets in the collection). To learn the knowledge of the Hitting Set Problem, DSL requires a look-up table to store all the possible combinations. DSL has been extended to support tasks with more complex perception requirements (Daniele et al., 2024), such as the CIFAR-10 dataset (Krizhevsky et al., 2009). In (Daniele et al., 2024), an Embed2Sym style approach is applied, where the perception component is trained using a fully neural method in an end-to-end fashion, before being frozen and inserted into a new architecture with the DSL component. Linear layers are used to learn an assignment from learned embedding representations to the input concepts to the DSL layer. Since this approach relies upon end-to-end training that is the same as the first stage of Embed2Sym (Aspis et al., 2022), it is unlikely to scale to tasks with a large amount of choice for the primitive concepts, as evidenced in our experiments in Chapter 4.

The DeepPSL and NeurPSL approaches (Dasaratha et al., 2023; Pryor et al., 2023a,b) extend the probabilistic soft logic reasoning framework, to support end-to-end neuro-symbolic learning. The reasoning step requires back-propagating gradients through a markov random field that does not have continuous derivatives. These approaches tackle this problem by developing a differentiable approximation. However, considering the rule learning, only the weights of the rules are learned, and a set of possible rules is defined manually, apriori. In contrast, our approaches are able to explore a search space of rules defined in terms of possible head and body relations, in order to learn general and expressive knowledge.

Finally, one approach not yet discussed, is (Evans et al., 2021). Instead of trying to make the symbolic component differentiable, so it can be integrated within a neural architecture, (Evans et al., 2021) does the exact opposite. They implement the neural component in ASP, to enable direct integration with the symbolic component. The weights of the network, and the rules of the task, are jointly learned within a single SAT problem. Since ASP does not support floating point arithmetic, (Evans et al., 2021) are restricted to training binary neural networks. In contrast, our approaches train general neural networks, and therefore can take advantage of GPU acceleration in the variety of deep-learning frameworks available.

7.2 Other Related Work

In this section, we review other related work within Neuro-Symbolic AI that is broadly related to the contributions presented in this thesis. We focus on works that train a neuro-symbolic architecture in the presence of a fixed symbolic knowledge base (called *neuro-symbolic reasoning*), since all of the contributions presented in this thesis aim to relax the requirement to specify the knowledge, and instead learn it using a symbolic learner. We also discuss work that leverages foundation models within a neuro-symbolic architecture, which is useful to compare to our NeSyGPT approach in Chapter 6 but also can be used for future work considerations. To leverage existing symbolic knowledge, there are many different approaches that can be used. Some approaches create neural network architectures that simulate the behaviour of logical formulae, whereas others train neural networks to extract primitive concepts from raw inputs, using the background knowledge that is either specified symbolically, or in a differentiable manner. There are also hybrid approaches that perform symbolic reasoning on top of pre-trained neural networks, and approaches that leverage foundation models to scale to complex tasks. In this section, we review the main techniques in each category.

7.2.1 Training Neural Networks to Simulate Logical Formulae

An early approach, called KBANN (Towell and Shavlik, 1994), translates (imperfect) symbolic background knowledge, expressed as acyclic, propositional Horn clauses, into a neural network architecture. The neurons in the network use a logistic activation function, so their output activations are in the range $[0, 1]$. The neurons are weighted to reflect the semantics of conjunction, disjunction or negation, and the idea is to apply weights such that the activations are close to 1, only if the corresponding deduction can be made using the background knowledge, and 0 otherwise. The (imperfect) background knowledge can then be refined through learning via back-propagation, where the weights are updated based on a set of training examples. It is also possible to add new neurons, that do not correspond to any propositions in the background knowledge, to refine the existing knowledge. Once the network is trained, the refined knowledge can then be extracted, and interpreted, since the neurons in the network map to a propositional structure. The KBANN idea was extended in the CILP system (Garcez and Zaverucha, 1999) by using logic programming as a knowledge representation language, instead of propositional Horn clauses. More recently, the idea of creating a 1-to-1 mapping between neurons and elements of a logical formulae, has been revisited in the Logical Neural Network (LNN) framework (Riegel et al., 2020), as discussed in Section 7.1. Recently, (Zhang et al., 2023b) propose the AutoInt method, which extends the Neuro-Rule-Network approach (Wang et al., 2020) to learn from continuous data. Neuro-Rule-Networks are similar to the LNN framework, as neural network architectures are constructed to simulate the behaviour of a logic program, and the activation function of neurons are modified to conform to the semantics of conjunction or disjunction. Neuro-Rule-Networks rely upon binarising input data through a one-hot encoding scheme, before the network performs the reasoning task. The AutoInt approach generalises the Neuro-Rule-Networks to accept continuous input data, by learning appropriate intervals to binarise the data.

Whilst these techniques have demonstrated improved performance compared to purely neural and purely symbolic systems, they are limited in terms of the expressivity of knowledge that can be represented. These architectures are limited to representing conjunction, disjunction, and negation, and do not support more advanced constructs such as recursion, general constraints, choice rules, or programs with multiple models, as learned by our systems presented in this

thesis. Also, these architectures have only been applied on symbolic datasets, and they are unable to accept high dimensional feature vectors as input, required in order to learn directly from raw data. The main drawback however, is that these systems require a strong background knowledge specification to define the network architecture and seed the learning process, and the learning is limited to refinement of existing knowledge. Our approaches on the other hand, explore a search space of possible rules, to learn general and highly expressive knowledge for solving complex problems.

7.2.2 Training Neural Networks with Logic Programs

DeepProbLog (Manhaeve et al., 2018) bridges the gap between the neural and symbolic components by interpreting neural network outputs as probabilistic atoms within the ProbLog framework (De Raedt et al., 2007). Specifically, DeepProbLog creates annotated disjunctions that represent the full choice of possible network outputs. The symbolic knowledge, also expressed in ProbLog, is then compiled into an arithmetic circuit, which can be used to train the network. To achieve this, a loss function calculates the gradient of the final output probability with respect to the downstream label, which can then be back-propagated through the circuit, until the gradients of each network output are calculated. From this point, standard neural network back-propagation updates the network weights. Since DeepProbLog places minimal restriction on the network architecture, it can be applied to perception and (probabilistic) reasoning tasks. The authors of DeepProbLog introduced the MNIST Arithmetic tasks used widely in the neuro-symbolic community, and they demonstrated DeepProbLog achieves strong results in this domain. However, since DeepProbLog relies upon calculating the full space of probabilities for each training example, during forward and backward inference, it becomes a challenge to scale to more complex tasks.

Along similar lines to DeepProbLog, NeurASP is able to train a neural network given a complete background knowledge expressed as an ASP program (Yang et al., 2020). NeurASP leverages choice rules in ASP, to simulate the annotated disjunctions used in DeepProbLog. To train the network, NeurASP then calculates all of the answer sets of an ASP program, which contains the choice rules, the background knowledge, and a constraint, to ensure only valid choices of network predictions are chosen that lead to the downstream label as output, given the background knowledge. NeurASP then calculates the gradients of the network outputs using the semantic loss function (Xu et al., 2018). NeurASP achieves comparable performance to DeepProbLog on the MNIST Addition task, but since NeurASP is based upon ASP, it can train the neural networks using a more expressive set of logical constraints. However, as discussed in Section 5, NeurASP relies upon calculating all of the answer sets of the ASP program, and ultimately suffers similar scalability issues to DeepProbLog. To address this, NeurASP has been extended in the SLASH framework (Skryagin et al., 2022), which prunes various answer sets during training. The authors observe the neural network gradients accumulate to certain concept values, long before the network reaches a high level of accuracy. This information is used to train the neural network with a more precise signal, which leads to faster convergence.

The Abductive Learning Framework (ABL) (Dai et al., 2019), and the numerous extensions (Huang et al., 2021; Cai et al., 2021; Huang et al., 2023; Yang et al., 2024; Tao et al., 2024) use a given background knowledge to abduce a set of pseudo-labels for training a neural network, given downstream task labels. The abduction process is similar to the answer set enumeration in NeurASP, except ABL identifies exactly one pseudo-label for training the network with a given raw input. The key challenge here is how to prune the full set of pseudo-labels to identify the

correct label that corresponds to each raw input in a particular example. The original framework relies upon the neural network confidence to select the most probable pseudo-labels that satisfy the downstream task. However, during the early stages of training, the neural network is not able to predict with high confidence, so the pseudo-label selection is highly dependant on the random initialisation of the network parameters. Therefore, numerous attempts have been made to prune the space of pseudo-labels for various tasks. For example, in natural language tasks, (Cai et al., 2021) use the string edit distance to bias towards pseudo-labels that better-match valid candidates within a dictionary of valid English words, whereas (Huang et al., 2021) use the notion of similarity to help identify a common set of pseudo-labels for raw images that have similar embeddings within a latent feature space. Also, (Huang et al., 2023) apply rule mining over a large knowledge graph to select only rules that are relevant to the task, which helps prune the (much larger) space of possible pseudo-labels that may otherwise be considered. Finally, (Yang et al., 2024) presents an approach to handle noisy background knowledge during the abduction process, using a graphical model that aims to minimise the discrepancy between the distribution of pseudo-labels generated by the neural network and the abduction process. This can be viewed as a pseudo-label correction technique in the presence of noisy rules.

NeuroLog (Tsamoura et al., 2021) also uses an abduction procedure to train the neural network, but instead considers *all* of the abductive solutions, in a similar manner to NeurASP and DeepProbLog, rather than a singular solution as in ABL. NeuroLog proposes a compositional perspective where the neural and symbolic modules are handled as black-boxes, with interfaces for neural induction and logical abduction, rather than being tied to a particular logical formalism. NeuroLog demonstrates superior performance compared to ABL, which the authors attribute to their consideration of all possible pseudo-labels. NeuroLog learns more efficiently than DeepProbLog and NeurASP, since NeuroLog caches abductive proofs to be used throughout training. However, computing the full abductive space remains an expensive operation, and NeuroLog, like the aforementioned systems, is unlikely to scale to complex problems.

Finally, there are a number of other approaches that utilise logic programs during neural network training (Andreas et al., 2016; Bošnjak et al., 2017; Mao et al., 2018; Li et al., 2023b). For example, in the domain of VQA, Neural Module Networks (Andreas et al., 2016) train neural networks with respect to a logic program extracted from natural language questions. The Differentiable Forth Interpreter (Bošnjak et al., 2017) is closely related to DeepProbLog, except it uses the programming language Forth as the symbolic component instead of ProbLog, so is unable to perform probabilistic reasoning. In (Mao et al., 2018), an iterative curriculum based learning approach is taken where neural networks are first trained to extract low-level concepts such as the colour and shape of objects, before being trained to answer more complex relational questions. In this system, called the Neuro-Symbolic Concept Learner, symbolic programs are extracted from questions which are used to train the neural network. The Scallop language (Li et al., 2023b) extends DeepProbLog by relaxing its exact semantics, and uses DataLog to increase the expressivity compared to ProbLog.

In contrast to our systems proposed in this thesis, all of these approaches require a complete background knowledge to be specified, whereas our approaches learn the knowledge. Our NSIL system leverages the NeurASP framework to achieve simultaneous training of both the neural network and the symbolic learner. However, approaches that require the knowledge to be expressed symbolically are challenged by scalability in complex tasks, since an appropriate training signal has to be determined by reasoning over the knowledge. Our FF-NSL and NeSyGPT systems avoid the scalability issues by decoupling the network training from symbolic learning and reasoning.

Another direction which aims to address the scalability issues are the approaches that represent the symbolic knowledge in a differentiable form, to enable seamless integration with end-to-end training of a neural network. Let us now discuss these approaches.

7.2.3 Training Neural Networks with Differentiable Knowledge

Logic Tensor Networks (LTNs) (Badreddine et al., 2022) have been proposed as an integrated learning and reasoning framework, to learn in the presence of logical constraints. LTNs use a first order logic syntax interpreted in the real numbers to represent symbolic knowledge, which together with feature vectors representing sub-symbolic knowledge, are implemented in a deep tensor network. Given the network architecture, LTNs enable feature vectors to be learned from data whilst hard logical constraints are satisfied. This is referred to as learning an approximate solution to the constraint optimisation problem called best satisfiability. Also, LTNs can be used to refine soft constraints in the presence of new data observations, given that logical constraints and feature vectors are implemented in the same deep tensor network architecture. LTNs have been applied to tasks such as knowledge completion (Badreddine et al., 2022), and semantic image interpretation (Donadello et al., 2017), where image object classifications are learned whilst inferring logical relations between image objects, using symbolic background knowledge. Despite LTNs enabling background knowledge to be refined given new data observations, they still suffer the limitation that new knowledge cannot be learned.

Another approach, called NeuroPlex (Xing et al., 2020), trains a neural network surrogate to simulate the behaviour of the symbolic knowledge. The training set is automatically generated, since the given knowledge can be used to compute target labels, given randomly generated input data. Once the surrogate network has been trained, its weights are frozen and the network is inserted into a fully differentiable pipeline for end-to-end training. For example, a perception neural network could be trained to extract concepts from raw images, using the downstream label and the frozen surrogate network. Whilst this is an effective approach, there is no guarantee the surrogate network perfectly approximates the symbolic knowledge, and generating an exhaustive training set may be difficult for complex problems.

Approaches such as (Cohen, 2016; Cohen et al., 2018; Rocktäschel and Riedel, 2017; Cohen et al., 2019) also focus on embedding symbolic knowledge into a neural network, to enable end-to-end differentiable architectures that are capable of answering natural language questions, learning from labelled question answer pairs and utilising background knowledge from an existing knowledge base. With these approaches, symbolic knowledge is embedded within a neural network using matrix and vector encodings such that the requirement for performing reasoning within a symbolic component is eliminated, and the system can be trained more efficiently due to the singular differentiable architecture. In the ASP domain, (Aspis et al., 2020) introduces a novel approach for computing stable and supported models of normal logic programs in continuous vector space, by one-hot encoding Herbrand interpretations as 0–1 vectors and representing program reducts as matrices. Whilst (Aspis et al., 2020) encodes expressive symbolic knowledge in the form of rules, constraints, facts and choices in a differentiable framework, the knowledge is still required apriori.

7.2.4 Symbolic Reasoning from Pre-Trained Neural Networks

Given that our FF-NSL approach utilises pre-trained neural networks, we briefly analyse neuro-symbolic reasoning approaches that also perform pre-training. These approaches use pre-trained neural components for extracting concepts, and they employ symbolic reasoning over the top of

neural network outputs. For example, the idea of integrating neural networks for learning, with symbolic components for reasoning has also been applied to the task of VQA (Yi et al., 2018; Aditya et al., 2018; Eiter et al., 2022), where an image is given alongside a natural language question about the image. In all of these approaches, pre-trained neural networks extract scene information from images, such as objects and their properties (e.g., shape, colour, size). In (Yi et al., 2018), LSTM networks are pre-trained to generate a sequential program of function calls required to answer a given natural language question as input. Then, logical rules that define a set of possible functions are manually implemented in a Python program. To answer a natural language question such as “*How many cubes that are behind the cylinder are large?*” given as input along with an image, scene information is first extracted using the pre-trained neural networks. Second, the question is converted into a sequential list of function calls using an LSTM and finally, the corresponding python functions are called to answer the question. In (Aditya et al., 2018), Word2Vec (Mikolov et al., 2013b) and ConceptNet (Speer et al., 2017) are used as background knowledge to capture concepts and relations between natural language words, and a semantic parser is used to extract nouns and adjectives from natural language questions. Probabilistic Soft Logic is used to reason over neural network outputs, the parsed question and background knowledge concepts, to return the most likely answer to a question.

(Eiter et al., 2022) use a pre-trained pipeline for performing VQA over the CLEVR dataset, where pre-trained neural networks extract objects and their properties. A natural language question is then encoded functionally as an ASP program, and the network predictions, as well as the ASP program can be solved to return the answer. This framework has also been extended in (Eiter et al., 2023) to generate explanations for the answers given, via an ASP reasoning task, and finally, applied to a graph dataset in (Bauer et al., 2023), where a pre-trained neural network extracts nodes in the graph given raw images as input.

Whilst these approaches apply neuro-symbolic reasoning to complex problems, all components of the architecture are manually engineered. In contrast, all of our systems learn the symbolic knowledge, and NSIL trains both the neural and symbolic components simultaneously.

7.2.5 Leveraging Foundation Models

Given the increasing success of large foundation models, there are also approaches that aim to leverage these models within a wider neuro-symbolic reasoning framework. LLMs in particular, excel at capturing and translating raw (user) input, and symbolic systems can perform reasoning, maintaining the benefits of a hybrid integration. Some approaches have demonstrated success in prompting LLMs to generate logic programs, and in particular, ASP programs (Yang et al., 2023; Ishay et al., 2023). This enables non-experts to define solutions to complex reasoning problems. In these techniques, the knowledge is given, albeit in a raw textual form. Other approaches use LLMs to semantically parse raw inputs, before performing downstream symbolic reasoning (Nye et al., 2021; Olausson et al., 2023). For example, (Nye et al., 2021) use the GPT-3 language model to generate candidate natural language sentences to complete a given story, before a symbolic reasoning system checks the consistency of this story, against a world model given as background knowledge. To achieve this, symbolic facts are extracted from the generated sentences, also using GPT-3. The system repeatedly samples generations from the language model, until one of the generations satisfies the given background knowledge. Along similar lines, (Olausson et al., 2023) use a LLM to translate natural language expressions to first-order logic, and they utilise a symbolic theorem prover to perform deductive inference, given the LLM

generations. Both (Nye et al., 2021) and (Olausson et al., 2023) demonstrate using this hybrid method is able to outperform purely neural systems that rely on LLMs only. However, there is still the question of where the symbolic world knowledge comes from, and, in certain domains, these approaches may require many iterations until a LLM generates satisfactory output.

Recently, the AlphaGeometry approach has achieved significant results when solving complex geometrical problems from the Math Olympiad (Trinh et al., 2024). This is a neuro-symbolic approach which combines a neural language model to suggest useful constructs to a symbolic theorem prover, which then computes the solution. Without the guidance of the language model, the space of possible solutions is intractable for the symbolic engine to compute solutions directly. However, the symbolic engine still relies upon a given background knowledge, so it is unclear how this approach could be applied in situations where the knowledge is unknown.

Finally, the STAR framework connects LLMs with ASP. STAR uses a LLM to generate ASP predicates (semantic parsing), before performing common-sense reasoning given a domain knowledge base. Again, there is a heavy reliance on the given knowledge base to solve the task.

Whilst the contributions presented in this thesis *learn* symbolic knowledge, recent approaches aim to leverage the large amount of world knowledge embedded within foundation models, to generate the symbolic knowledge. For example, (Wang et al., 2023) use a LLM to perform program induction, and ask the LLM to generate hypotheses at multiple levels of abstraction. These hypotheses are then down-selected, by using another LLM query to summarise the generated hypotheses into a smaller set. (Wang et al., 2023) also discuss using human annotators to perform this step. Finally, the LLM translates the down-selected hypotheses into Python code, which are evaluated on the training examples. The hypothesis that covers the most examples is then returned as the inductive solution. The authors evaluate their approach on the challenging abstract reasoning corpus, and achieve improved results compared to a naive baseline that uses the LLM to generate the hypothesis single-shot. Whilst these results are promising, there is no guarantee the LLM will generate an optimal solution, and the code generation step relies upon a significant amount of training data in the form of Python code. As we discovered in our experiments in Chapter 6, it currently seems challenging to use a LLM to generate ASP programs zero-shot, which may be required for tasks where it is more natural to express solutions in a declarative manner. In contrast, the symbolic learners used in our systems are guaranteed to learn an optimal solution with respect to their given inputs.

(Zhang et al., 2023a) introduce an end-to-end neuro-symbolic framework that leverages a LLM to generate relational probabilistic facts, given an input natural language paragraph. For example, assuming the input “Alice is the daughter of Bob”, the language model generates a relational fact `daughter(Alice,Bob)`, with a certain probability. The goal is to predict an output relation, which is achieved using the Scallop differentiable reasoning system (Li et al., 2023b), that contains weighted rules. The predicted output is then compared to the ground-truth relation, and both the language model, and scallop are trained end-to-end via the semantic loss. Instead of manually engineering the rules, (Zhang et al., 2023a) use a language model to generate them, and then tune their weights through end-to-end training. Whilst this is an effective approach, it is not clear whether the framework can handle other types of raw data such as imagery, since it is only applied to natural language inputs. Also, there is no guarantee the language model will generate the correct set of rules, and therefore there is no optimality guarantee, unlike the symbolic learners used in this thesis.

In the visual domain, (Surís et al., 2023) demonstrate using GPT-4 to generate Python programs to solve various perception and reasoning tasks. They rely upon external pre-trained modules to perform the perception tasks, such as segmentation, object detection, and VQA. (Surís

et al., 2023) prompts GPT-4 to perform tool-use, where the function signatures, and documentation strings, are given, along with the user’s input query. GPT-4 is then tasked with combining the various “tools” together to solve the input query. This significantly extends the domain of tasks GPT-4 can be applied to. However, there is no guarantee GPT-4 will generate a correct program, and the Python representation used may not be suitable for tasks that are of a declarative nature.

7.2.6 Summary

In this chapter, we have reviewed the relevant literature that is directly related to our contributions within the area of neuro-symbolic learning, as well as approaches that are broadly related within the areas of neuro-symbolic reasoning, and recent techniques that leverage foundation models. Neuro-symbolic reasoning approaches assume the symbolic knowledge is given, and the focus is on training the neural perception component to extract primitive concepts from raw data. When considering the directly related work where the knowledge is learned, existing approaches are either restricted in the expressivity of knowledge that can be learned, are difficult to scale to complex tasks, provide no formal guarantees on the optimality of the learned knowledge, or are restricted to training binary neural networks. In contrast, all of our approaches are able to learn expressive knowledge such as negation, choice, constraints, and programs with multiple models, and also train general neural network architectures. Furthermore, our FF-NSL and NeSyGPT approaches provide formal guarantees of learning optimal symbolic learning, with respect to the noisy predictions from the neural network. In the next chapter, we summarise the contributions of this thesis and outline directions for future work.

Chapter 8

Conclusion

In this chapter, we summarise the contributions of this thesis, and outline directions for future work.

8.1 Summary of Contributions

We have explored and developed four approaches for integrating neural networks with symbolic learning systems, all of which learn general and interpretable knowledge from raw data. In doing so, we have introduced new benchmark tasks and datasets of the desired complexity, in order to showcase the benefits of our integrated neuro-symbolic systems, and serve as a useful resource for the neuro-symbolic community.

First, our FF-NSL approach integrates *pre-trained* neural networks to extract primitive concepts from raw data. Neural network predictions, and confidence scores, are transformed into weighted examples for the symbolic learner, using a t -norm aggregation function to compute the example weight. The challenge with using a pre-trained neural network, is the network may be unreliable, since the data for the task at hand may originate from a different distribution than the data used for pre-training. The symbolic learner is then faced with noisy examples, where noise occurs in the accuracy of the concept predictions, and also the example weights. To the best of our knowledge, this is the first study to evaluate the performance of symbolic learning systems when given neural network predictions from raw input data that are subject to distributional shifts. Our comprehensive evaluation demonstrates FF-NSL is able to learn accurate knowledge, even in cases where a large percentage of the training examples are subject to distributional shifts. Since the symbolic learner is able to generalise effectively, by balancing the length of the learned hypothesis with the example coverage, it can leverage a few accurate examples within the learning task to learn the correct knowledge. FF-NSL outperforms other hybrid integrations, where a neural network and a Random Forest are used as the symbolic learning component. However, since FF-NSL assumes a pre-trained neural network, the approach requires training with a large labelled dataset, if no pre-trained network is available. This is cumbersome in practice, since a sufficiently large dataset may be difficult, or expensive to obtain.

Therefore, our second contribution, NSIL, relaxes the requirement for the neural network to be pre-trained, and both the neural and symbolic components are trained simultaneously, where no labels are required for the primitive concepts. Such an integration is challenging, since the symbolic learner is not differentiable, and therefore a standard end-to-end approach cannot apply. We solve this challenge using a novel iterative approach, whereby a candidate

hypothesis is learned, and the neural network is trained using the NeurASP neuro-symbolic reasoning framework. Since the initial candidate hypothesis is likely to be incorrect, the trained neural network is also likely to be sub-optimal. We overcome this, and complete the iterative loop by using *corrective examples*, which bias the symbolic learner to either refine or retain the current hypothesis for the next iteration. The corrective examples are computed using the FNR, which is indicative of the overall system performance, as well as the neural network predictions and confidence scores. The corrective examples represent positive and negative cases for a given combination of concepts and a downstream task label, where the positive (resp. negative) case indicates that the label should be (resp. should not be) output for the given concepts. NSIL effectively balances exploration vs. exploitation of the hypothesis search space. To the best of our knowledge, this is the first system capable of learning expressive ASP programs, whilst simultaneously training a neural network to extract primitive concepts from raw data. Our evaluation demonstrates NSIL is able to learn the correct hypothesis, and train an accurate neural network in a variety of perception and reasoning tasks. Also, NSIL outperforms fully neural, and neuro-symbolic baselines, including the closest competitor, *Meta_{Abd}*, achieving state-of-the-art performance.

Despite this, NSIL faces two main challenges. First, NSIL is not guaranteed to converge to an optimal hypothesis, since the iterative adjustment of the corrective examples can result in NSIL oscillating between hypotheses. Second, in the context of a perception and reasoning task, it is difficult to scale NSIL, and other end-to-end approaches in general, since there is a combinatorial explosion in the interface between the neural and symbolic components. Specifically, it becomes challenging to train the neural network when there is a large amount of choice for the values of the primitive concepts, since these ultimately have to be abduced given a symbolic hypothesis and a downstream task label. This leads to a weak gradient signal shared between many possibilities. This challenge exists in neuro-symbolic reasoning (which NSIL relies upon), and becomes harder in neuro-symbolic learning, where the hypothesis also has to be learned. As a result, our third contribution explores a novel, decoupled approach for neuro-symbolic learning. In this architecture, the neural network is trained independently using self-supervised learning, and the symbolic learner is tasked with learning a hypothesis given learned representations, or clusters, of the raw input data. In developing novel approaches along this direction, we ultimately faced scalability challenges, since there remains a combinatorial issue of assigning the correct values to the primitive concepts, even if the perception component is able to be trained independently. Also, this approach is restricted to image classification tasks, since it relies upon a clustering of the raw input data in an embedding space. It is not clear whether tasks with more advanced perception requirements, such as object detection, could be supported.

Our final contribution therefore revisits the idea of sequential training, to solve the scalability challenges encountered in end-to-end systems, whilst learning optimal hypotheses with respect to the neural network predictions. The challenge is, can we develop a scalable and robust architecture, that is able to learn with less labelled data, and manual engineering than other approaches? To achieve this, we leverage recent developments within vision-language foundation models, and integrate VQA fine-tuning with a symbolic learner, in an architecture we call NeSyGPT. The novelty is transforming the natural language answers predicted from the VQA perception component into training examples for the symbolic learner. We evaluate NeSyGPT on a variety of challenging tasks, with a large amount of choice for the possible values of the primitive concepts. We achieve state-of-the-art performance compared to relevant neural and neuro-symbolic baselines that are capable of reasoning or learning in the language of ASP, whilst learning with very few labelled data points. NeSyGPT also requires a reduced amount of engineering, since the

VQA component is more general than a CNN, as it does not require bespoke tuning for a specific task. Furthermore, we demonstrate using a LLM to reduce the amount of manual engineering required to “glue” the neural and symbolic components together, as the LLM can generate the questions and answers for fine-tuning BLIP, as well as the Python code that transforms the perception predictions into symbolic training examples. Finally, we demonstrate NeSyGPT achieves state-of-the-art results on a perception and reasoning task where object detection is required. NeSyGPT outperforms the next-best baseline, α ILP on the challenging CLEVR-Hans task.

8.2 Future Work

To extend the work presented in this thesis, there are numerous directions one could take. These are outlined in this section.

Neural Interpretability Whilst we have focused on learning interpretable symbolic rules, there is an outstanding challenge to interpret what the neural network has learned. Although our architectures rely upon an intermediary concept layer, which enables the network to be probed with various samples to gain an understanding of cases where certain concept predictions are made, there is an open question as to *why* the network is predicting certain concepts, both globally, and locally, for a specific set of inputs. Work along this direction includes saliency maps (Simonyan et al., 2013), which highlight salient regions in image inputs that are important for model predictions. Saliency maps largely target CNN architectures, so one could easily apply these techniques to the perception networks in FF-NSL and NSIL. Interpreting more advanced architectures, such as the vision transformer, and joint vision and language models, is the focus of a large amount of effort within the AI community. For example, (Aflalo et al., 2022) is a recent approach that provides visualisations for the attention mechanisms within multi-modal transformers, that explains which features the transformer is attending to. This approach could be utilised by the BLIP model, which we leveraged in our NeSyGPT approach in Chapter 6. However, end-to-end interpretability of a neuro-symbolic architecture remains an outstanding challenge, since current neural interpretability components are “add-ons” that can be leveraged after training. It would be interesting to develop an integrated approach, that aims to explain why certain downstream predictions have been made, based upon learned neural features, and also the symbolic rules. LLMs could potentially be used to help explain the symbolic component in a more user-friendly manner, assuming it is also possible to quantify why certain neural network predictions are made, in natural language.

Scalable Symbolic Learning One of the major bottlenecks of all the contributions of this thesis is the scalability of symbolic machine learning. The LAS systems have achieved significant improvements compared to the previous state-of-the-art, and can now learn optimal rules in very large search spaces (Law, 2018; Law et al., 2020a). However, due to the combinatorial explosion of the hypothesis space, given a linear increase in the predicates defined in the mode declarations, this ultimately poses a challenge. In practice, one may wish to express an extremely large search space, especially if the target hypothesis is unknown (otherwise the hypothesis may as well be engineered manually). Whilst it is difficult to see any immediate gains along this direction, one may wish to revisit the idea with the advent of quantum computing. Along more pragmatic lines, one could also explore the use of a LLM to generate the search space for the symbolic learner. Here, the general world knowledge of the LLM could be leveraged, and the symbolic learner can focus on learning an optimal hypothesis in a smaller search space.

Learning the Number of Primitive Concepts Although in Chapter 6 we argue against pursuing an end-to-end architecture, since it becomes more difficult to control what is learned, it would nevertheless be interesting to explore whether a scalable end-to-end architecture could be developed that didn’t require *any* specification of the intermediary concept layer. Specifically, it would be interesting to explore whether the assumption of the number of primitive concepts could be removed. Current neuro-symbolic architectures require this, which ultimately requires the user to have a strong assumption and knowledge over what the concepts are. In this case, one might as well use our NeSyGPT approach, where very few labelled data points are required to fine-tune the perception component. Approaches along this direction could explore whether a LLM could be used to suggest candidate concept sets (Oikarinen et al., 2023), or whether a symbolic learner could discover the correct concept set, given an even broader specification than what is assumed today. However, if an LLM could suggest the concepts, this itself suggests that the concepts are known, at least in the public domain, since the training data for the LLM must have come from somewhere.

Scaling up VQA Finally, the BLIP vision-language foundation model used in Chapter 6 is fairly small compared to other models such as GPT-4 Vision. Another direction for future work could be to add fine-tuning to the larger foundation models, which in-turn would benefit the performance of NeSyGPT, without requiring any modification to our framework, or input from the user. As these models become more capable, and learn from larger datasets, it may be possible to use the models in a zero-shot fashion, without any fine-tuning. Currently, as demonstrated in our evaluation, fine-tuning is required in order to achieve an acceptable level of performance.

Bibliography

- M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenekov, and S. Nahavandi. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021. ISSN 1566-2535. URL <https://www.sciencedirect.com/science/article/pii/S1566253521001081>.
- S. Aditya, Y. Yang, and C. Baral. Explicit reasoning over end-to-end neural architectures for visual question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- E. Aflalo, M. Du, S.-Y. Tseng, Y. Liu, C. Wu, N. Duan, and V. Lal. VI-interpret: An interactive visualization tool for interpreting vision-language transformers. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pages 21406–21415, 2022.
- J. Andreas, M. Rohrbach, T. Darrell, and D. Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48, 2016.
- Y. Aspis, K. Broda, A. Russo, and J. Lobo. Stable and supported semantics in continuous vector spaces. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning*, 2020.
- Y. Aspis, K. Broda, J. Lobo, and A. Russo. Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, 8 2022.
- Y. Aspis, M. Albinhassan, J. Lobo, and A. Russo. Embed2rule scalable neuro-symbolic learning via latent space weak-labelling. In *Proceedings of International Conference on Neural-Symbolic Learning and Reasoning*, pages 195–218. Springer, 2024.
- S. Badreddine, A. d’Avila Garcez, L. Serafini, and M. Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022. ISSN 0004-3702. URL <https://www.sciencedirect.com/science/article/pii/S0004370221002009>.
- P. Baldi and P. J. Sadowski. Understanding dropout. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf.
- J. J. Bauer, T. Eiter, N. H. Ruiz, and J. Oetsch. Neuro-symbolic visual graph question answering with llms for language parsing. In *TAASP 2023, Workshop on Trends and Applications of Answer Set Programming, November 20-21, 2023, Potsdam, Germany, 2023*.

- C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *Proceedings of International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.
- R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.
- M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel. Programming with a differentiable forth interpreter. In *Proceedings of International conference on machine learning*, pages 547–556. PMLR, 2017.
- S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf.
- A. Byerly, T. Kalganova, and I. Dear. No routing needed between capsules. *Neurocomputing*, 463:545–553, 2021.
- L.-W. Cai, W.-Z. Dai, Y.-X. Huang, Y.-F. Li, S. Muggleton, and Y. Jiang. Abductive learning with ground knowledge base. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1815–1821. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/250. URL <https://doi.org/10.24963/ijcai.2021/250>. Main Track.
- T. Charalambous, Y. Aspis, and A. Russo. Neurafastlas: Fast logic-based learning from raw data. *arXiv preprint arXiv:2310.05145*, 2023.
- H. Chen, J. Yang, C. Vondrick, and C. Mao. Interpreting and controlling vision foundation models via text explanations. *arXiv preprint arXiv:2310.10591*, 2023.
- T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020.
- N. Cingillioglu. *End-to-end neuro-symbolic learning of logic-based inference*. PhD thesis, Imperial College London, 2022.
- K. L. Clark. Negation as failure. In *Logic and data bases*, pages 293–322. Springer, 1977.
- W. W. Cohen. Tensorlog: A differentiable deductive database. *CoRR*, abs/1605.06523, 2016. URL <http://arxiv.org/abs/1605.06523>.
- W. W. Cohen, F. Yang, and R. K. Mazaitis. Tensorlog: Deep learning meets probabilistic databases. *Journal of Artificial Intelligence Research*, 1:1–15, 2018.

- W. W. Cohen, H. Sun, R. A. Hofer, and M. Siegler. Scalable neural methods for reasoning with a symbolic knowledge base. In *International Conference on Learning Representations*, 2019.
- E. COM. Laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts. *Proposal for a regulation of the European parliament and of the council*, 2021.
- D. Corapi, A. Russo, and E. Lupu. Inductive logic programming as abductive search. In *Technical Communications of the 26th International Conference on Logic Programming*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.
- P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Wine quality. UCI Machine Learning Repository, 2009.
- A. Cropper and S. Dumančić. Inductive logic programming at 30: a new introduction. *Journal of Artificial Intelligence Research*, 74:765–850, 2022.
- D. Cunningham, M. Law, J. Lobo, and A. Russo. Ffns1: feed-forward neural-symbolic learner. *Machine Learning*, 112(2):515–569, 2023a.
- D. Cunningham, M. Law, J. Lobo, and A. Russo. Neuro-symbolic learning of answer set programs from raw data. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 3586–3596, 2023b.
- D. Cunningham, M. Law, J. Lobo, and A. Russo. The role of foundation models in neuro-symbolic learning and reasoning. In T. R. Besold, A. d’Avila Garcez, E. Jimenez-Ruiz, R. Confalonieri, P. Madhyastha, and B. Wagner, editors, *Neural-Symbolic Learning and Reasoning*, pages 84–100, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-71167-1.
- W.-Z. Dai and S. Muggleton. Abductive knowledge induction from raw data. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1845–1851. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.
- W.-Z. Dai, Q. Xu, Y. Yu, and Z.-H. Zhou. Bridging machine learning and logical reasoning by abductive learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/9c19a2aa1d84e04b0bd4bc888792bd1e-Paper.pdf.
- A. Daniele, T. Campari, S. Malhotra, and L. Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3597–3605. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.
- A. Daniele, T. Campari, S. Malhotra, and L. Serafini. Simple and effective transfer learning for neuro-symbolic integration. In *Proceedings of International Conference on Neural-Symbolic Learning and Reasoning*, pages 166–179. Springer, 2024.
- S. Dasaratha, S. A. Puranam, K. S. Phogat, S. R. Tiyyagura, and N. P. Duffy. Deeppsl: End-to-end perception and reasoning. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3606–3614. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.

- L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI 2007, Proceedings of the 20th international joint conference on artificial intelligence*, pages 2462–2467. IJCAI-INT JOINT CONF ARTIF INTELL, 2007.
- C. Demir and A.-C. N. Ngomo. Neuro-symbolic class expression learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 3624–3632, 2023.
- A. P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, 30(2):205–247, 1968. ISSN 00359246. URL <http://www.jstor.org/stable/2984504>.
- L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.
- J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- I. Donadello, L. Serafini, and A. D. Garcez. Logic tensor networks for semantic image interpretation. *arXiv preprint arXiv:1705.08968*, 2017.
- A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- T. Eiter, N. Higuera, J. Oetsch, and M. Pritz. A neuro-symbolic asp pipeline for visual question answering. *Theory and Practice of Logic Programming*, 22(5):739–754, 2022.
- T. Eiter, T. Geibinger, N. H. Ruiz, and J. Oetsch. A logic-based approach to contrastive explainability for neurosymbolic visual question answering. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI 2023)*, 2023.
- R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.
- R. Evans, M. Bošnjak, L. Buesing, K. Ellis, D. Pfau, P. Kohli, and M. Sergot. Making sense of raw input. *Artificial Intelligence*, 299:103521, 2021.
- K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
- D. Furelos-Blanco. *Learning and Exploiting Reward Machines for Reinforcement Learning*. PhD thesis, Imperial College London, 2023.
- A. Gadetsky, Y. Jiang, and M. Brbic. Let go of your labels with unsupervised transfer. *arXiv preprint arXiv:2406.07236*, 2024.
- A. d. Garcez and L. C. Lamb. Neurosymbolic ai: The 3 rd wave. *Artificial Intelligence Review*, 56(11):12387–12406, 2023.
- A. d. Garcez, S. Bader, H. Bowman, L. C. Lamb, L. de Penning, B. Illuminoo, H. Poon, and C. G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *Neuro-Symbolic Artificial Intelligence: The State of the Art*, 342(1):327, 2022.

- A. S. A. Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence*, 11(1):59–77, 1999.
- M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.
- M. Gelfond and Y. Kahl. *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, 2014.
- M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080. Cambridge, MA, 1988.
- I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- M. M. Gupta and J. Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy sets and systems*, 40(3):431–450, 1991.
- R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1735–1742, 2006.
- S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.
- J. Haugeland. *Artificial intelligence: The very idea*. MIT press, 1989.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.
- P. Hitzler and M. K. Sarker. Neuro-symbolic artificial intelligence: The state of the art. *Frontiers in Artificial Intelligence and Applications*, 342, 2022.
- Y.-X. Huang, W.-Z. Dai, L.-W. Cai, S. H. Muggleton, and Y. Jiang. Fast abductive learning by similarity-based consistency optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 34, pages 26574–26584. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/df7e148cabfd9b608090fa5ee3348bfe-Paper.pdf.
- Y.-X. Huang, Z. Sun, G. Li, X. Tian, W.-Z. Dai, W. Hu, Y. Jiang, and Z.-H. Zhou. Enabling abductive learning to exploit knowledge graph. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 3839–3847, 2023.
- D. P. Hughes and M. Salathé. An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. *CoRR*, abs/1511.08060, 2015. URL <http://arxiv.org/abs/1511.08060>.
- E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

- D. Hutchins, I. Schlag, Y. Wu, E. Dyer, and B. Neyshabur. Block-recurrent transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 35, pages 33248–33261. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/d6e0bbb9fc3f4c10950052ec2359355c-Paper-Conference.pdf.
- M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.
- A. Ishay, Z. Yang, and J. Lee. Leveraging large language models to generate answer set programs. *arXiv preprint arXiv:2307.07699*, 2023.
- A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.
- E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=rkE3y85ee>.
- N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous multivariate distributions*, volume 7. Wiley New York, 1972.
- A. Jøssang. *Subjective logic*, volume 3. Springer, 2016.
- D. Kahneman. *Thinking, fast and slow*. macmillan, 2011.
- R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
- A. Karpathy. Intro to large language models, 2023. URL https://www.youtube.com/watch?v=zjkBMFhNj_g&t=2100s.
- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- E. P. Klement, R. Mesiar, and E. Pap. *Triangular norms*, volume 8. Springer Science & Business Media, 2013.
- P. W. Koh, T. Nguyen, Y. S. Tang, S. Mussmann, E. Pierson, B. Kim, and P. Liang. Concept bottleneck models. In *Proceedings of International Conference on Machine Learning*, pages 5338–5348, 2020.
- T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8bb0d291acd4acf06ef112099c16f326-Paper-Conference.pdf.
- S. Kotz, N. Balakrishnan, and N. L. Johnson. *Continuous multivariate distributions, Volume 1: Models and applications*, volume 334. John wiley & sons, 2019.

- M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.
- A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.
- A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
- R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher. *Computational Intelligence: A Methodological Introduction*. Springer Nature, 2022.
- H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- H. Lakkaraju, S. H. Bach, and J. Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1675–1684, 2016.
- M. Law. *Inductive learning of answer set programs*. PhD thesis, Imperial College London, 2018.
- M. Law, A. Russo, and K. Broda. Inductive learning of answer set programs. In *Logics in Artificial Intelligence: 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings 14*, pages 311–325. Springer, 2014.
- M. Law, A. Russo, and K. Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Systems*, 7:57–76, 08 2018.
- M. Law, A. Russo, E. Bertino, K. Broda, and J. Lobo. Fastlas: scalable inductive logic programming incorporating domain-specific optimisation criteria. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2877–2885, 2020a.
- M. Law, A. Russo, and K. Broda. The ilasp system for inductive learning of answer set programs. *arXiv preprint arXiv:2005.00904*, 2020b.
- Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- D. Li, J. Li, H. Le, G. Wang, S. Savarese, and S. C. Hoi. Lavis: A library for language-vision intelligence. *arXiv preprint arXiv:2209.09019*, 2022a.
- J. Li, D. Li, C. Xiong, and S. Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *Proceedings of International Conference on Machine Learning*, pages 12888–12900. PMLR, 2022b.
- J. Li, D. Li, S. Savarese, and S. Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023a.

- Y. Li, M. Yang, D. Peng, T. Li, J. Huang, and X. Peng. Twin contrastive learning for online clustering. *International Journal of Computer Vision*, 130(9):2205–2221, 2022c.
- Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.
- Z. Li, J. Huang, and M. Naik. Scallop: A language for neurosymbolic programming. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1463–1487, 2023b.
- Z. Li, Y. Huang, Z. Li, Y. Yao, J. Xu, T. Chen, X. Ma, and J. Lu. Neurosymbolic learning yielding logical constraints. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 36, pages 21635–21657. Curran Associates, Inc., 2023c. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/4459c3c143db74ee52afebdf56836375-Paper-Conference.pdf.
- V. Lifschitz. *Answer set programming*, volume 3. Springer Heidelberg, 2019.
- J. W. Lloyd. *Foundations of logic programming*. Springer Science & Business Media, 2012.
- A. López-Cifuentes, M. Escudero-Viñolo, J. Bescós, and Álvaro García-Martín. Semantic-aware scene recognition. *Pattern Recognition*, 102:107256, 2020. ISSN 0031-3203. URL <https://www.sciencedirect.com/science/article/pii/S0031320320300613>.
- D. J. C. Mackay. Probable networks and plausible predictions — a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6(3):469–505, 1995.
- L. Mahon and T. Lukasiewicz. Selective pseudo-label clustering. In *Proceedings of Advances in Artificial Intelligence: 44th German Conference on AI, Virtual Event, September 27–October 1, 2021, Proceedings 44*, pages 158–178. Springer, 2021.
- R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. Deep-problog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.
- J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *Proceedings of International Conference on Learning Representations*, 2018.
- E. Marconato, S. Teso, A. Vergari, and A. Passerini. Not all neuro-symbolic concepts are created equal: Analysis and mitigation of reasoning shortcuts. *Proceedings of Advances in Neural Information Processing Systems*, 36, 2024.
- A. Margeloiu, M. Ashman, U. Bhatt, Y. Chen, M. Jamnik, and A. Weller. Do concept bottleneck models learn as intended? *ICLR 2021 Workshop on Responsible AI*, 2021.
- W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

- G. Metcalfe, N. Olivetti, and D. M. Gabbay. *Proof theory for fuzzy logics*, volume 36. Springer Science & Business Media, Berlin/Heidelberg, Germany, 2008.
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.
- T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *Proceedings of International Conference on Learning Representations*, 2013b.
- I. Misra and L. v. d. Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6707–6717, 2020.
- C. Molnar. *Interpretable Machine Learning*. Lean Publishing, Online, 2019.
- S. Muggleton. Inductive logic programming. *New generation computing*, 8:295–318, 1991.
- S. Muggleton. Inverse entailment and prolog. *New generation computing*, 13:245–286, 1995.
- C. Niu, H. Shan, and G. Wang. Spice: Semantic pseudo-labeling for image clustering. *IEEE Transactions on Image Processing*, 31:7264–7278, 2022.
- V. Novák, I. Perfilieva, and J. Mockor. *Mathematical principles of fuzzy logic*, volume 517. Springer Science & Business Media, 2012.
- M. Nye, M. Tessler, J. Tenenbaum, and B. M. Lake. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 34, pages 25192–25204. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/d3e2e8f631bd9336ed25b8162aef8782-Paper.pdf.
- T. Oikarinen, S. Das, L. M. Nguyen, and T.-W. Weng. Label-free concept bottleneck models. *arXiv preprint arXiv:2304.06129*, 2023.
- T. Olausson, A. Gu, B. Lipkin, C. Zhang, A. Solar-Lezama, J. Tenenbaum, and R. Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176, Singapore, Dec. 2023. Association for Computational Linguistics. URL <https://aclanthology.org/2023.emnlp-main.313>.
- OpenAI. Gpt-4 system card - openai, 2023. URL <https://cdn.openai.com/papers/gpt-4-system-card.pdf>.
- OpenAI, :, J. Achiam, and e. a. Steven Adler. Gpt-4 technical report, 2023.
- L. C. Paulson. Computational logic: its origins and applications. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2210):20170872, 2018.
- T. Pearce, A. Brintrup, and J. Zhu. Understanding softmax confidence and uncertainty, 2021. *arXiv:2106.04972*.

- C. Pryor, C. Dickens, E. Augustine, A. Albalak, W. Y. Wang, and L. Getoor. Neupsl: Neural probabilistic soft logic. In E. Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 4145–4153. International Joint Conferences on Artificial Intelligence Organization, 8 2023a. Main Track.
- C. Pryor, C. Dickens, and L. Getoor. Deep neuro-symbolic weight learning in neural probabilistic soft logic. In *ICML Workshop on Knowledge and Logical Reasoning in the Era of Data-Driven Learning (KLR)*, 2023b.
- A. Quattoni and A. Torralba. Recognizing indoor scenes. In *2009 IEEE conference on computer vision and pattern recognition*, pages 413–420. IEEE, 2009.
- A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *Proceedings of International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- C. E. Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.
- R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020.
- R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.
- T. Rocktäschel and S. Riedel. End-to-end differentiable proving. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 3791–3803, 2017.
- F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- S. Russell and P. Norvig. A modern approach. *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, 90:33–48, 2002.
- P. Sen, B. W. S. R. d. Carvalho, R. Riegel, and A. Gray. Neuro-symbolic inductive logic programming with logical neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8):8212–8219, Jun. 2022. URL <https://ojs.aaai.org/index.php/AAAI/article/view/20795>.
- M. Sensoy, L. Kaplan, and M. Kandemir. Evidential deep learning to quantify classification uncertainty. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/a981f2b708044d6fb4a71a1463242520-Paper.pdf.

- M. Sensoy, L. Kaplan, F. Cerutti, and M. Saleki. Uncertainty-aware deep classifiers using generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5620–5627, 2020.
- H. Shindo, M. Nishino, and A. Yamamoto. Differentiable inductive logic programming for structured examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5034–5041, 2021.
- H. Shindo, V. Pfanschilling, D. S. Dhami, and K. Kersting. α ilp: thinking visual scenes as differentiable logic programs. *Machine Learning*, 112(5):1465–1497, 2023.
- K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of 3rd International Conference on Learning Representations*, pages 1–14. Computational and Biological Learning Society, 2015.
- K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.
- D. Singh, N. Jain, P. Jain, P. Kayal, S. Kumawat, and N. Batra. Plantdoc: A dataset for visual plant disease detection. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, pages 249–253, 2020.
- A. Skryagin, W. Stammer, D. Ochs, D. S. Dhami, and K. Kersting. Neural-probabilistic answer set programming. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 463–473, 2022.
- A. Skryagin, D. Ochs, D. S. Dhami, and K. Kersting. Scalable neural-probabilistic answer set programming. *arXiv preprint arXiv:2306.08397*, 2023.
- R. Speer, J. Chin, and C. Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 4444–4451, 2017. URL <http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14972>.
- W. Stammer, P. Schramowski, and K. Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3619–3629, 2021.
- D. Surís, S. Menon, and C. Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 11888–11898, 2023.
- M. S. Tanveer, M. U. K. Khan, and C.-M. Kyung. Fine-tuning darts for image classification. In *Proceedings of 25th International Conference on Pattern Recognition*, pages 4789–4796. IEEE, 2021.
- L. Tao, Y.-X. Huang, W.-Z. Dai, and Y. Jiang. Deciphering raw data in neuro-symbolic learning with provable guarantees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 15310–15318, 2024.
- G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1-2):119–165, 1994.
- T. H. Trinh, Y. Wu, Q. V. Le, H. He, and T. Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, 2024.

- E. Tsamoura, T. Hospedales, and L. Michael. Neural-symbolic integration: A compositional perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5051–5060, 2021.
- G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.
- L. G. Valiant. Three problems in computer science. *Journal of the ACM (JACM)*, 50(1):96–99, 2003.
- A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Proceedings of Advances in Neural Information Processing Systems*, 30, 2017.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.
- R. Wang, E. Zelikman, G. Poesia, Y. Pu, N. Haber, and N. D. Goodman. Hypothesis search: Inductive reasoning with language models. *arXiv preprint arXiv:2309.05660*, 2023.
- Z. Wang, W. Zhang, L. Ning, and J. Wang. Transparent classification with multilayer logical perceptrons and random binarization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 6331–6339, 2020.
- A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/81b3833e2504647f9d794f7d7b9bf341-Paper.pdf.
- T. Xing, L. Garcia, M. R. Vilamala, F. Cerutti, L. Kaplan, A. Preece, and M. Srivastava. Neuroplex: learning to detect complex events in sensor networks through knowledge injection. In *Proceedings of the 18th conference on embedded networked sensor systems*, pages 489–502, 2020.
- J. Xu, Z. Zhang, T. Friedman, Y. Liang, and G. Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of International conference on machine learning*, pages 5502–5511. PMLR, 2018.
- X.-W. Yang, J.-J. Shao, W.-W. Tu, Y.-F. Li, W.-Z. Dai, and Z.-H. Zhou. Safe abductive learning in the presence of inaccurate rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16361–16369, 2024.
- Z. Yang, A. Ishay, and J. Lee. Neurasp: Embracing neural networks into answer set programming. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

- Z. Yang, A. Ishay, and J. Lee. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*, 2023.
- K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. Tenenbaum. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *Proceedings of Proceedings of Advances in Neural Information Processing Systems*, pages 1031–1042, 2018.
- L. Yujian and L. Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.
- H. Zhang, J. Huang, Z. Li, M. Naik, and E. Xing. Improved logical reasoning of language models via differentiable symbolic programming. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the Association for Computational Linguistics*, pages 3062–3077, Toronto, Canada, July 2023a. Association for Computational Linguistics. URL <https://aclanthology.org/2023.findings-acl.191>.
- W. Zhang, Y. Liu, Z. Wang, and J. Wang. Learning to binarize continuous features for neuro-rule networks. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 4584–4592, 2023b.

Appendix A

LAS Task Specifications

Listing A.1: Follow Suit Winner.

```
% Background knowledge
suit(h). suit(s). suit(d). suit(c). rank(a). rank(2).
rank(3). rank(4). rank(5). rank(6). rank(7). rank(8).
rank(9). rank(10). rank(j). rank(q). rank(k).
rank_value(2, 2). rank_value(3, 3). rank_value(4, 4).
rank_value(5, 5). rank_value(6, 6). rank_value(7, 7).
rank_value(8, 8). rank_value(9, 9). rank_value(10, 10).
rank_value(j, 11). rank_value(q, 12). rank_value(k, 13).
rank_value(a, 14). player(1..4).

rank_higher(P1, P2) :- card(P1, R1, _), card(P2, R2, _),
    rank_value(R1, V1), rank_value(R2, V2), V1 > V2.

% Link player's card to suit
suit(P1, S) :- card(P1, _, S).

% Mode declarations to specify hypothesis search space
P(X) :- Q(X), identity(P, Q).
P(X) :- player(X), not Q(X), inverse(P, Q).
#mode(2, inverse(target/1, invented/1)).
#mode(2, identity(target/1, invented/1)).
#predicate(target, winner/1).
#predicate(invented, p1/1).
#modeh(p1(var(player))).
#modeb(1, var(suit) != var(suit)).
#modeb(1, suit(var(player), var(suit)), (positive)).
#modeb(1, suit(const(player), var(suit)), (positive)).
#modeb(1, rank_higher(var(player), var(player)), (positive)).
#constant(player, 1).
#constant(player, 2).
#constant(player, 3).
#constant(player, 4).
```

Listing A.2: Sudoku 4x4 Grid Validity.

```
% Background knowledge
row(1..4).
col(1..4).
block(1..4).
cell(C) :- value(C, _).
neq(X, Y) :- cell(X), cell(Y), X != Y.
num(1..4).

% Link cell identifiers to cell values
value("1, 1", X) :- value(1, 1, X).
value("1, 2", X) :- value(1, 2, X).
value("1, 3", X) :- value(1, 3, X).
```

```

value("1, 4", X) :- value(1, 4, X).
...
value("4, 1", X) :- value(4, 1, X).
value("4, 2", X) :- value(4, 2, X).
value("4, 3", X) :- value(4, 3, X).
value("4, 4", X) :- value(4, 4, X).

% Columns
col("1, 1", 1).
col("1, 2", 2).
col("1, 3", 3).
col("1, 4", 4).
...
col("4, 1", 1).
col("4, 2", 2).
col("4, 3", 3).
col("4, 4", 4).

% Rows
row("1, 1", 1).
row("1, 2", 1).
row("1, 3", 1).
row("1, 4", 1).
...
row("4, 1", 4).
row("4, 2", 4).
row("4, 3", 4).
row("4, 4", 4).

% Blocks
block("1, 1", 1).
block("1, 2", 1).
block("2, 1", 1).
block("2, 2", 1).
...
block("3, 3", 4).
block("3, 4", 4).
block("4, 3", 4).
block("4, 4", 4).

% Mode declarations to specify hypothesis search space
#modeh(invalid).
#modeb(value(var(cell), var(num))).
#modeb(row(var(cell), var(row))).
#modeb(not row(var(cell), var(row))).
#modeb(col(var(cell), var(col))).
#modeb(not col(var(cell), var(col))).
#modeb(block(var(cell), var(block))).
#modeb(not block(var(cell), var(block))).
#modeb(neq(var(cell), var(cell))).
#maxv(4).
#bias("penalty(1, head).").
#bias("penalty(1, body(X)) :- in_body(X).").

```

Listing A.3: Sudoku 4x4 Grid Validity when explicit knowledge about the grid is removed.

```

% Background knowledge
div_same1(X, Y, C) :- (X - 1) / C = (Y - 1) / C, idx1(X), idx1(Y), X < Y, quotient(C).
div_same2(X, Y, C) :- (X - 1) / C = (Y - 1) / C, idx2(X), idx2(Y), X < Y, quotient(C).

quotient(1..3).
idx1(1..4).
idx2(1..4).
num(1..4).

% Mode Declarations
#modeh(invalid).
#modeb(value(var(idx1), var(idx2), var(num))).

```

```
#modeb(div_same1(var(idx1), var(idx1), const(quotient))).
#modeb(div_same2(var(idx2), var(idx2), const(quotient))).
#maxv(5).

#bias("penalty(1, head).").
#bias("penalty(1, body(X)) :- in_body(X).").
#ground_without_replacement.
```

Listing A.4: Crop Yield Prediction.

```
% Background knowledge
:- yield(X), yield(Y), X < Y.
yield_type(0..2).

% Mode declarations
#maxv(4).
#modeh(yield(const(yield_type))).
#modeb(1, field(const(field))).
#modeb(1, species(const(species))).
#modeb(1, disease(const(disease))).

#bias("penalty(1, head).").
#bias("penalty(1, body(X)) :- in_body(X).").
```

Listing A.5: Indoor Scene Classification.

```
% Background Knowledge
label_type(0..4).

% Mode Declarations
#modeh(label(const(label_type))).
#modeb(1,image(const(image))).

#bias("penalty(1, head).").
#bias("penalty(1, body(X)) :- in_body(X).").
```

Listing A.6: Cumulative Arithmetic.

```
% List definition in ASP
(L) :- start_list(L).
(T) :- list(_, T).
(L, H) :- list(L, L = (H, _).
(L, T) :- list(L, L = (_, T).
y(empty).

ithmetic Knowledge - add, mult, eq
L, (X+Y, T) :- list(L, L = (X, (Y, T)).
(L) :- add(_, L).
(L, (X*Y, T)) :- list(L, L = (X, (Y, T)).
(L) :- mult(_, L).
, ELT) :- list(L, L = (ELT, empty).

nk learned f to result, ensure only one result
lt(R) :- start_list(L), f(L, R).
esult(X), result(Y), X < Y.

dicate(base, head/2).
dicate(base, tail/2).
dicate(base, add/2).
dicate(base, mult/2).
dicate(base, eq/2).
dicate(base, empty/1).
dicate(target, f/2).

ta rules
B) :- Q(A, B), m1(P, Q).
B) :- Q(A, C), P(C, B), m2(P, Q).
B) :- Q(A, C), R(C, B), m3(P, Q, R), Q != R.
```

```

B) :- Q(A, B), R(A, B), m4(P, Q, R), Q != R.
:- Q(A, B), m5(P, Q, B).
:- Q(A), m6(P, Q).
B) :- Q(A), R(A, B), m7(P, Q, R).
B) :- Q(A, B), R(B), m8(P, Q, R).

em(2, m1(target/2, any/2)).
em(2, m2(target/2, any/2, any/2)).
em(3, m3(target/2, any/2, any/2)).
em(3, m4(target/2, any/2, any/2)).
em(2, m5(target/1, any/2)).
em(2, m6(target/1, any/1)).
em(3, m7(target/2, any/1, any/2)).
em(3, m8(target/2, any/2, any/1)).

```

Listing A.7: Two Digit Arithmetic.

```

% Background Knowledge
even(X) :- digit_type(X), X \ 2 = 0.
plus_nine(X1,X2) :- digit_type(X1), X2=9+X1.
:- digit(1,X0), digit(2,X1), result(Y1), result(Y2), Y1 != Y2.
result(Y) :- digit(1,X0), digit(2,X1), solution(X0,X1,Y).
num(0..18).
digit_type(0..9).

% Mode Declarations
#modeh(solution(var(digit_type),var(digit_type),var(num))).
#modeb(var(num) = var(digit_type)).
#modeb(var(num) = var(digit_type) + var(digit_type)).
#modeb(plus_nine(var(digit_type),var(num))).
#modeb(even(var(digit_type))).
#modeb(not even(var(digit_type))).
#maxv(3).

#bias("penalty(1, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").

```

Listing A.8: Hitting Set Problem.

```

% Background Knowledge
ss(1..4).
hs_index(1..2).
elt(1..5).

% Meta-Level Injection
#inject("
1 { example_active(EG) : group(G, EG) } :- group(G, _).
:- not example_active(EG), weight(EG, W).[W@1, eg_weight, W, EG]
:- #count { H : nge_HYP(H) } > 5.
").

% Mode Declarations
#modeha(hs(var(hs_index), var(elt))).
#modeb(hs(var(hs_index), var(elt)), (positive)).
#modeb(var(elt) != var(elt)).
#modeb(ss_element(var(ss), var(elt)), (positive)).
#modeh(hit(var(ss))).
#modeb(hit(var(ss))).
#bias(":- not lb(0), choice_rule.").
#bias(":- not ub(1), choice_rule.").
#bias(":- in_head(H1), in_head(H2), H1<H2.").

#modeb(ss_element(3, var(elt)), (positive)).
#modeb(ss_element(var(ss), 1), (positive)).

```

Listing A.9: Plant Hitting Sets.

```

% Background Knowledge

```

```

ss(1..5). hs_index(1..2). elt(1..38). healthy(4).
healthy(5). healthy(7). healthy(11). healthy(15).
healthy(18). healthy(20). healthy(23). healthy(24).
healthy(25). healthy(28). healthy(38).

% Mode Declarations
#modeha(hs(var(hs_index), var(elt))).
#modeb(hs(var(hs_index), var(elt)), (positive)).
#modeb(var(elt) != var(elt)).
#modeb(ss_element(var(ss), var(elt)), (positive)).
#modeh(hit(var(ss))).
#modeb(hit(var(ss))).
#modeb(healthy(var(elt)), (positive)).
#bias(":- not lb(0), choice_rule.").
#bias(":- not ub(1), choice_rule.").
#bias(":- in_head(H1), in_head(H2), H1<H2.").

```

Listing A.10: CLEVR-HANS.

```

% Background Knowledge
:- label(X), label(Y), X < Y. label_type(0..2).
obj_id(1..10). s_type(small). s_type(large).
m_type(rubber). m_type(metal). c_type(red).
c_type(cyan). c_type(brown). c_type(yellow).
c_type(blue). c_type(gray). c_type(purple).
c_type(green). sh_type(cylinder). sh_type(sphere).
sh_type(cube).

% Mode Declarations
#maxv(2).
#modeh(label(const(label_type))).
#modeb(size(var(obj_id), const(s_type))).
#modeb(material(var(obj_id), const(m_type))).
#modeb(color(var(obj_id), const(c_type))).
#modeb(shape(var(obj_id), const(sh_type))).
#bias("penalty(1, head(X)) :- in_head(X).").
#bias("penalty(1, body(X)) :- in_body(X).").

```


Appendix B

NeSyGPT Example Generators

For the tasks considered in the NeSyGPT evaluation in Chapter 6, this section details the python code listings for the example generators. The example generator for the Two-Digit Arithmetic task is shown in Listing 6.1 in the main body of the thesis. Let us now present the example generators for the other tasks.

```

def generate_example(
    example_id: int,
    blip_predictions: list[dict[str, str]],
    label: str,
    meta_data: list | None = None
) -> WCDPI:
    """
    This function generates a WCDPI for the Follow Suit Winner task.
    Args:
        example_id (int): The identifier for the example.
        blip_predictions (list[dict[str, str]]): A list of predictions from BLIP. The list contains
            a dictionary of questions and answers related to the concepts in each image.
        label (str): The target task label.
        meta_data (list, optional): The optional meta-data associated with the data point.
    Returns (WCDPI):
        A WCDPI object representing the training example for the symbolic learner.
    """
    # Create the example context using the BLIP predictions for each image.
    context: list[str] = []
    for image_index, image_predictions in enumerate(blip_predictions):
        rank = image_predictions["which rank is this?"]
        if rank in ["jack", "queen", "king", "ace"]:
            rank = rank[0]
        suit = image_predictions["which suit is this?"][0]
        player = meta_data[image_index].split("Player ")[1]
        context.append(f"card({player},{rank},{suit}).")

    # Construct the WCDPI object.
    return WCDPI(
        ex_id=f"ex_{example_id}",
        positive=True,
        weight=10,
        inclusion=[f"winner({label})"],
        exclusion=[
            f"winner({1})"
            for l in range(1, NUM_PLAYERS + 1)
            if l != label
        ],
        context=context
    )

```

Listing B.1: Example Generator for the Follow Suit Winner Task.

```

def generate_example(
    example_id: int,
    blip_predictions: list[dict[str, str]],
    label: str,
    meta_data: list | None = None
) -> WCDPI:
    """
    This function generates a WCDPI for the Plant Hitting Sets task.
    Args:
        example_id (int): The identifier for the example.
        blip_predictions (list[dict[str, str]]): A list of predictions from BLIP. The list contains
            a dictionary of questions and answers related to the concepts in each image.
        label (str): The target task label.
        meta_data (list, optional): The optional meta-data associated with the data point.
    Returns (WCDPI):
        A WCDPI object representing the training example for the symbolic learner.
    """
    # Create the example context using the BLIP predictions for each image.
    context: list[str] = []

    # Get the set identifiers for each image (defined externally). SET_IDENTIFIERS is a list
    # of pipe-separated integers denoting the set ID for each image. For example,
    # "1|1|2|2|3" denotes a collection of three sets, where the first and second images are in
    # set 1, the third and fourth images are in set 2, and the fifth image is in set 3.
    set_ids = SET_IDENTIFIERS[example_id].split("|")

    # For each image prediction, create a contextual fact.
    for image_index, image_predictions in enumerate(blip_predictions):
        species = image_predictions["which plant species is this?"]
        disease = image_predictions["which disease is this?"]

        # Try looking up the class identifier based on the predicted species and disease. If we
        # can't, pick a random disease associated with this species.
        try:
            class_id = CLASS_ID_LOOKUP[f"{species}_{disease}"]
        except KeyError:
            new_disease = random.choice(DISEASE_LOOKUP[species])
            class_id = CLASS_ID_LOOKUP[f"{species}_{new_disease}"]

        # Add a contextual fact for this image.
        context.append(
            f"ss.element({set_ids[image_index]},{class_id})."
        )

    # Construct the WCDPI object.
    return WCDPI(
        ex_id=f"ex_{example_id}",
        positive=int(label) == 1,
        weight=10,
        inclusion=[],
        exclusion=[],
        context=context
    )

```

Listing B.2: Example Generator for the Plant Hitting Sets Task.

```

def generate_example(
    example_id: int,
    blip_predictions: list[dict[str, str]],
    label: str,
    meta_data: list | None = None
) -> WCDPI:
    # Create the example context using the BLIP predictions for the image.
    context: list[str] = []

    # Extract the individual object predictions from an answer such as "The image contains a
    # large gray cube made of metal, a large yellow..."
    answer = blip_predictions[0]["What does the image contain?"]
    objects = answer.split("The image contains ")[1]
    objects = objects.split(", ")

    for object_index, clevr_object in enumerate(objects):
        # Remove "made of", the leading "a ", and "."
        clevr_object = clevr_object.replace("made of", "")
        clevr_object = clevr_object.replace(".", "")
        clevr_object = clevr_object[2:]

        # Split the object string into attributes.
        attributes = clevr_object.split(" ")
        object_id = object_index + 1

        # For each attribute, create the corresponding contextual fact.
        for a in attributes:
            if a in ["small", "large"]:
                context.append(f"size({object_id}, {a}).")
            elif a in ["metal", "rubber"]:
                context.append(f"material({object_id}, {a}).")
            elif a in ["red", "cyan", "brown", "yellow",
                       "blue", "gray", "purple", "green"]:
                context.append(f"color({object_id}, {a}).")
            elif a in ["cylinder", "cube", "sphere"]:
                context.append(f"shape({object_id}, {a}).")
            else:
                raise ValueError(f"Unknown Attribute: {a}")

        # Construct the WCDPI object.
        exclusions = [
            f"label({1})"
            for l in range(3)
            if str(l) != label
        ]
    return WCDPI(
        ex_id=f"ex_{example_id}",
        positive=True,
        weight=1,
        inclusion=[f"label({label})"],
        exclusion=exclusions,
        context=context
    )

```

Listing B.3: Example Generator for the CLEVR-Hans3 Task.