Symbolic Discovery in Continuous Space with End-to-End Differentiability

Kexin Gu

Department of Computing
Imperial College London

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

July 2026

Statement of Originality

I, Kexin Gu (also known as Kexin Gu Baugh), hereby declare that this thesis represents my original work. All work from other sources has been properly cited and acknowledged. A full list of bibliography is provided at the end of this thesis.

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

Neural networks achieve remarkable success across domains, yet their black-box nature poses challenges in safety-critical applications. Neuro-symbolic approaches address this by integrating symbolic inference with deep learning to enhance interpretability. Many works follow the principle of differentiable inductive logical programming, but typically require prior knowledge to define or restrict the search space, thus limiting expressivity and applicability. The neural Disjunctive Normal Form (DNF) model stands out as a fully end-to-end differentiable model that provides integration with neural components for predicate invention from raw data and logical interpretation via post-training rule extraction. Yet, its effectiveness is only demonstrated in synthetic binary classification tasks. This thesis proposes a family of neural DNF-based models that perform fully differentiable rule learning across classification and reinforcement learning (RL) tasks. These models retain the interpretability of the original and are shown to be amenable to behaviour modification via weight editing.

This thesis presents four main contributions. Firstly, it formalises the behaviour and interpretation of neural DNF-based models, establishing a bidirectional translation with truth-value equivalence guaranteed between neural and bivalent logic forms at post-training, supporting model interpretation and intervention.

Secondly and thirdly, the thesis explores the application of neural DNF-based models in classification and actor-critic RL settings, which impose different behavioural constraints: classification requires logically deterministic predictions and mutual exclusivity for multi-class problems, while RL actors should output probabilistic mutually exclusive action distributions. In both setups, the models perform on par with competing methods while providing compact, interpretable, and intervenable logical forms.

Finally, the thesis investigates performance degradation when translating trained neural DNF-based models into logical forms. It identifies a knowledge entanglement problem encoded in models' weights, and proposes a disentanglement solution that splits nodes encoding nested rules into smaller, independent ones. Experiments show that this method better preserves models' representation, achieving performance closer to pre-translation levels.

So, whether you eat or drink, or whatever you do, do all to the glory of God.
1 Corinthians 10:31

Acknowledgements

First and foremost, I wish to thank my supervisors and mentors for their guidance, encouragement and unwavering support, all of which have been invaluable in helping me grow as a researcher. Alessandra Russo has been a wonderful supervisor ever since the final year of my MEng degree and throughout my PhD. Her passion for the neuro-symbolic field and dedication to her students have been truly inspiring. Her logic-based AI course served as an excellent introduction to the symbolic AI field for me in my third year of the MEng degree, leading me to undertake an MEng project with her on neuro-symbolic learning (back when LLMs/VLMs were not yet a thing and I had to stack a bunch of different neural components to extract structural representations from unstructured data). With her advice and mentorship, I have now contributed to the neuro-symbolic field through my publications and research. I am forever grateful for her introducing me to this field and for her steadfast guidance over the years. Luke Dickens joined as a co-supervisor in my second year of my PhD and provided invaluable direction and encouragement. His strong background in deep learning proved to be a tremendous asset, and he was instrumental in helping me formalise my ideas and research. He also pushed me to think more deeply about both the immediate and broader implications of my work in neuro-symbolic AI, while helping me correct some of my less effective research practices. I have learned a great deal from him about writing and presenting my research, and I am deeply indebted to him for all his assistance. Katsumi Inoue was a great supervisor during my research exchange at the National Institute of Informatics in Tokyo. I appreciate his mentorship and his connections to the Japanese AI community as well as logicians and researchers worldwide. The mini-conferences I attended while in Japan were wonderful opportunities to learn, although I could not understand some of the talks in Japanese. I am also grateful to Nuri Cingillioglu for his mentorship during my final year of my MEng and my first year of my PhD. I jokingly 'blame' Nuri for 'dragging' me into the world of neuro-symbolic AI with his lunchtime talk when I was in the third year of my MEng degree. We often joked about how much of a 'repeat' my journey was compared to his Imperial years (MEng and then straight to PhD with Ale). Without his introduction and encouragement, I would not have pursued a PhD in the first place, let alone be where I am today.

I am thankful to my examiners, Luis Lamb and Francesco Belardinelli, for their careful reading of this thesis and the stimulating discussion during the viva. Their insightful feedback and questions have helped sharpen the ideas presented in this work.

I also wish to thank all the SPIKE members and alumni for their friendship and support over the years. In particular, I am grateful to Alex F. Spies, Charles Pert, Daniel Furelos-Blanco, Frederik Kelbel, Hadeel Al-Negheimish, Leo Ardon, Mohammad Albinhassan, Nuri Cingillioglu, Paweł Gomoluch, Rakhilya Lee Mekhtieva and Roko Parac. There were many interesting research discussions, alongside not-so-useful random chats, memes and drawings of fat pigeons and Pokémon on whiteboards. The intense SPIKE esports gaming sessions were also 'relaxing' and helped recharge my brain after long days of research. Beyond the SPIKE members, I wish to thank Vincent Perreault, with whom I worked at NII, for his inspiration and companionship during my time in Japan. His strong mathematical background proved invaluable in helping me formalise the last piece of the puzzle: the disentanglement method presented in Chapter 7 of this thesis.

Finally, I wish to thank my family, both here in the UK and in China, for their love and understanding throughout this journey. Without them, I would not be where I am today. My parents, Jun Zhao and Tao Gu, despite being far away in China, have always been there for me

via video calls and messages, and their interest in my work and the AI field in general has been a constant source of encouragement. My in-laws, Elizabeth and John Baugh, have offered tremendous physical and emotional support, reminding me to maintain a healthy work-life balance even during the more 'chaotic' periods when I needed to meet tight deadlines. Last but not least, I wish to thank my husband, Matthew Baugh. I am so glad we had the opportunity to write and publish papers together. Thank you for always being there for me and for all the sparks of creativity and peace of mind you have given me. Whenever I was worried and anxious, you were there to remind me how God has been sustaining me all along and I am safe in His hands.

Contents

List of Figures xiii

List of Tables xvii

Acronyms xxi

Notation xxiii

1 Introduction 1
- 1.1 Contributions . . . 3
  - 1.1.1 Theoretical Foundations . . . 3
  - 1.1.2 Neural DNF-based Models for Classification and Reinforcement Learning . . . 4
  - 1.1.3 Model Discretisation: from Continuous Space to Discrete Space . . . 4
- 1.2 Publications . . . 5
- 1.3 Thesis Structure . . . 6

2 Background 7
- 2.1 Answer Set Programming . . . 7
- 2.2 Deep Learning . . . 9
  - 2.2.1 Neural Networks . . . 10
  - 2.2.2 Optimisation . . . 11
  - 2.2.3 Regularisation . . . 12
  - 2.2.4 Classification Tasks in Deep Learning . . . 13
- 2.3 Semi-Symbolic Node and Neural DNF Model . . . 16
  - 2.3.1 Model Formulation . . . 17
  - 2.3.2 Training . . . 18
  - 2.3.3 Post-training Process for Logical Interpretation . . . 19
- 2.4 Reinforcement Learning . . . 21
  - 2.4.1 Fundamentals . . . 21
  - 2.4.2 Actor-Critic Methods . . . 23

3 Related Work 27
- 3.1 Neuro-symbolic Learning . . . 27
  - 3.1.1 End-to-end Rule Learning . . . 27
  - 3.1.2 Rule Learning with Hybrid Architectures . . . 34

3.1.3 Neural-Symbolic Predicate Invention . . . 36
3.1.4 Network Learning with Logical Constraints . . . 37
3.2 Neuro-symbolic Reinforcement Learning . . . 37
3.2.1 Policy Learning for RL . . . 38
3.2.2 Hybrid Models with Classic ILP for RL . . . 40

4 Semi-Symbolic Nodes and Neural DNF Model 43
4.1 Formalisation of Semi-Symbolic Node . . . 43
4.1.1 Interpretation and Soft-Valued Truth Table . . . 44
4.1.2 Discretisation . . . 45
4.1.3 Properties of Discretised Semi-Symbolic Node . . . 45
4.2 Bidirectional Neural-Bivalent-Logic Translation . . . 48
4.2.1 Symbolisation and Projection . . . 49
4.2.2 Neural-to-Bivalent-Logic Translation with Truth Value Equivalence . . . 50
4.2.3 Bivalent-Logic-to-Neural Translation with Truth Value Equivalence . . . 57

5 Neural DNF-based Models for Classification Tasks 63
5.1 Neural DNF-EO Model . . . 64
5.1.1 The Challenge of Multi-class Classification . . . 64
5.1.2 Model Architecture . . . 65
5.1.3 Post-training Process . . . 66
5.2 Experiments . . . 67
5.2.1 Real-world Multi-class Task: CUB Dataset/Subsets . . . 68
5.2.2 Real-world Multi-label Task: TMC Dataset/Subsets . . . 74
5.2.3 Synthetic Dataset . . . 77
5.3 Discussion . . . 79
5.3.1 Classification Performance . . . 79
5.3.2 Interpretability . . . 80
5.3.3 Scalability . . . 80
5.4 Conclusion . . . 81

6 Neural DNF-MT Model for Reinforcement Learning 83
6.1 Issues of Existing Neural DNF-based Models . . . 84
6.2 Methodology . . . 85
6.2.1 Mutex-tanh Activation and Neural DNF-MT . . . 85
6.2.2 Architecture and Training of Neural DNF-MT as Actor with PPO . . . 86
6.2.3 Post-training Processing . . . 88
6.2.4 Neural-bivalent-logic translation for Deterministic Policy Interpretation and Policy Intervention . . . 90
6.3 Experiments . . . 90
6.3.1 Switcheroo Corridor . . . 91
6.3.2 Blackjack . . . 93
6.3.3 Taxi . . . 95
6.3.4 Door Corridor . . . 96

6.3.5 Policy Intervention in Door Corridor Variants . . . 99
6.4 Discussions . . . 100
6.4.1 Performance Loss due to Thresholding . . . 100
6.4.2 Implications of Neural DNF-MT Model . . . 101
6.5 Conclusion . . . 103

7 The Entanglement Problem and Disentanglement **105**
7.1 The Problem of Entanglement . . . 106
7.1.1 Underemphasising . . . 106
7.1.2 Overemphasising . . . 107
7.2 Exploration of Stronger Logical Bias: `Semi-Symbolic-MinBias` Node . . . 107
7.2.1 `Semi-Symbolic-MinBias`, a Stricter Logical Constraint . . . 108
7.2.2 Limitations of `Semi-Symbolic-MinBias` . . . 110
7.3 Disentanglement . . . 111
7.3.1 Disentangling a Positively Connected Conjunctive Node . . . 111
7.3.2 Disentangling a Negatively Connected Conjunctive Node . . . 113
7.3.3 Optimisation . . . 115
7.3.4 Model Sparsification . . . 126
7.4 Additional Learning Techniques . . . 126
7.4.1 Predicate Invention . . . 126
7.4.2 Improved Delta Scheduling . . . 127
7.5 Experiments . . . 130
7.5.1 Setup . . . 130
7.5.2 Results . . . 132
7.6 Discussions . . . 133
7.6.1 Limitations when Disentangling Disjunctive Layers . . . 136
7.6.2 Scalability of Disentanglement . . . 136
7.7 Conclusion . . . 137

8 Conclusion and Future Work **139**
8.1 Summary of Contributions . . . 139
8.2 Future Work . . . 140
8.2.1 Model Extensions . . . 140
8.2.2 Applications in Real-life Domains . . . 142

Bibliography **145**

A Semi-Symbolic Node and Neural DNF Model: Supplementary Material **161**
A.1 Properties of Discretised Semi-Symbolic Nodes . . . 161
A.2 Proofs of Neural-Bivalent-Logic Translation . . . 162

B Classification Experiments **171**
B.1 CUB Dataset/subsets . . . 171
B.1.1 Data Pre-processing . . . 171
B.2 TMC Dataset/subsets . . . 171

B.2.1 Data Pre-processing 171
B.3 Logical Interpretation Examples 173

C Neural DNF-MT Model for Reinforcement Learning **183**
C.1 Post-training Algorithms 183
C.2 Switcheroo Corridor 187
C.3 Blackjack 189
C.4 Taxi 192
C.5 Door Corridor 198

D Proofs of Disentanglement Method **201**

E Code Repositories **207**

List of Figures

1.1 An illustration of an original learned continuous hypothesis in blue and a hypothesis after some degree of discretisation in orange. The discrete targets that the hypothesis is expected to cover are marked with dots (with yellow or red dashed lines connected to them). The discretisation process shrinks the area of coverage by pushing the peaks tighter around the discrete targets, and the area of the difference between the original and hypothesis after discretisation is shaded in blue. However, it also changes the behaviour of the function. If the value is interpreted as true when the hypothesis is above the black dashed line, then there is a discrete target marked in red that was covered by the original hypothesis in blue but is now missed by the new hypothesis in orange. 5

2.1 Comparison of the activation functions and their derivatives. 11

2.2 Comparison of L_p regularisation functions. 13

2.3 A neural DNF model with a conjunctive layer followed by a disjunctive layer. Conjunctive nodes in the conjunctive layer have their δ values greater than 0, while disjunctive nodes in the disjunctive layer have their δ values less than 0. 17

2.4 An example of a delayed exponential decay schedule of δ, using the calculation in Equation (2.7) with parameters $\delta_0 = 0.1$, $r = 1.1$, $t_{\text{delay}} = 5$ and $t_{\text{steps}} = 2$. 18

2.5 The post-training process to extract ASP rules from a neural DNF model. 19

2.6 The agent-environment interaction in an MDP [150]. 22

4.1 The auxiliary loss function $|w||6 - |w||$ used to encourage weights to converge to the discrete set $\{-6, 0, 6\}$. The function reaches zero at the desired discrete values and has a maximum penalty at $w = \pm 3$. Compared to L_1 regularisation, $|w||6 - |w||$ also allows ± 6 to be used, enforcing more than just sparsity. 46

5.1 An example of a neural DNF-EO model. 66

5.2 The post-training pipelines used in this chapter's experiments to extract ASP rules from a neural DNF-EO model (b) and from a neural DNF model (c), in comparison to the post-training process (a) in Section 2.3.3. 67

5.3 A sample from the CUB-200-2011 dataset before any data pre-processing. The image is ignored and the input to the neural DNF-based models contains the attributes only. 69

5.4 Average rule length and maximum rule length comparison across rules learned from FastLAS, extracted rules from trained neural DNF-EO model and decision tree (both pruned and without pruning). The pruned decision trees still maintain an accuracy of 80% in training sets. FastLAS runs out of memory for CUB-200. 73

5.5 Evaluation of the decision tree's performance on TMC-22 dataset with different maximum depths. The red dotted line represents 8000 number of non-leaf nodes that is similar to the number of parameters in an MLP/neural DNF model. Both the macro F1 and the number of nodes decrease as the depth decreases. When the decision tree's maximum depth is limited to 15, it has 2,810 non-leaf nodes and achieves a test macro F1 of 0.565, still better than the MLP's performance but close to it. 78

5.6 An example of an unstable equilibrium. 80

6.2 Post-training processing to extract an interpretable logical policy from a trained neural DNF-MT actor. There are two branches: one with sub-label (a) for extracting a stochastic policy in ProbLog and the other with sub-label (b) for extracting a deterministic policy in ASP. 88

6.3 Mean episodic return ($\pm$ standard error) across all environments for baseline methods, neural DNF-MT models, and the ProbLog/ASP programs extracted from the trained neural DNF-MT models. Higher values indicate better performance. Q-tables are trained using Q-learning, while MLP and neural DNF-MT actors are trained using actor-critic PPO with MLP critics. In the Taxi environment only, the neural DNF-MT actor is distilled from a pre-trained MLP actor (indicated by dashed borders and faded colouring). Symbols denote action selection methods: * argmax selection, † ϵ-greedy sampling, ‡ sampling from the actor's distribution. 90

6.4 The three environments in Switcheroo Corridor Environment Set. 91

6.5 Policy grid of a neural DNF-MT actor in the Blackjack environment without post-training processing. Each square shows the probability of taking action 'Hit' and the argmax action in the text. If a square is red, it means that its argmax action is different from the baseline optimal policy from [150]. The redder the square, the larger the probability difference. 94

6.6 Extracted ProbLog policy grid of the same neural DNF-MT actor in Figure 6.5. . . . 94

6.7 Taxi Environment: the taxi needs to pick up a passenger and drop them off at the destination hotel to finish the episode. 95

6.8 Door Corridor (DC): the agent needs to turn right first, and toggle and go through three doors to reach the end of the corridor. 97

6.9 Time taken vs the number of annotated disjunctions in ProbLog, using a ProbLog program in a certain format as an approximation. The overall trend is that the time taken increases exponentially with the number of rules presented in the ProbLog program. However, this pattern cannot be confirmed as the same for inference with an actual policy extracted from a neural DNF-MT actor. 103

7.2 The three steps of the disentanglement method. Here w is the weight connecting conjunctive node c to disjunctive node d. 112

7.3 Split elements in the original conjunctive node's weights $\mathbf{w}$ into two sets, based on the absolute values of the elements. The original sign of the weights is preserved in the figure, where the purple weights are positive and the yellow weights are negative. The set on the left is the small-magnitude weights where $|w_j| < \max|\mathbf{w}|/2$, and the set on the right is the large-magnitude weights where $|w_j| \geq \max|\mathbf{w}|/2$. $\bar{\mathcal{J}}^+$ is the set of indices correspond to the purple weights in the set on the left, and $\bar{\mathcal{J}}^-$ is the set of indices correspond to the yellow weights in the set on the left. $\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+$ is the set of indices of the weights in the set on the right that are positive (purple), and $\mathcal{J}^- \setminus \bar{\mathcal{J}}^-$ is the set of indices of the weights in the set on the right that are negative (yellow). All the weights in the set on the right should be present in all the split weight vectors in the split weight set $\mathcal{W}$. 118

7.4 Split elements in the original conjunctive node's weights $\mathbf{w}$ into two sets, based on the absolute values of the elements. The original sign of the weights is preserved in the figure, where the purple weights are positive and the yellow weights are negative. The set on the left is the small-magnitude weights where $|w_j| < \max|\mathbf{w}|/2$, and the set on the right is the large-magnitude weights where $|w_j| \geq \max|\mathbf{w}|/2$. $\bar{\mathcal{J}}^+$ is the set of indices correspond to the purple weights in the set on the left, and $\bar{\mathcal{J}}^-$ is the set of indices correspond to the yellow weights in the set on the left. $\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+$ is the set of indices of the weights in the set on the right that are positive (purple), and $\mathcal{J}^- \setminus \mathcal{J}^-$ is the set of indices of the weights in the set on the right that are negative (yellow). All the weights in the set on the right alone are sufficient to make the original conjunctive node's behaviour false. Each weight in the set on the right can be the only element present in a split weight vector in the split weight set $\mathcal{W}$. 123

7.5 Architectures for various settings with different types of inputs, where blue rounded boxes have learnable parameters. 7.5a is the basic setup for bivalent inputs. 7.5b and 7.5c show how predicate invention is performed with MLPs and threshold learning. . 127

7.6 Examples of new delayed delta scheduling methods with different update functions and different settings. 129

7.7 An example of the delayed monitoring linear delta scheduling using $f_{\delta,\text{delayed monitoring linear}}$ from Equation 7.26. The schedule halts at $|\delta| = 0.6$ before continuing to increase. . . 131

7.8 Comparison of different delayed delta scheduling methods during training on the CDC and Covertype datasets. The accuracy curves are shown as solid lines and the delta magnitude curves are shown as dashed lines. The figures show the stability benefits of different update functions over exponential updates. 132

7.9 Time taken (blue) to disentangle the conjunctive nodes with different numbers of inputs used, and the number of split conjunctions (red) after disentanglement, in a trained neural DNF model. A max runtime of 180s is set for disentanglement. 137

8.1 A proposal of a multi-layer neural DNF-based model with three neural DNF blocks. Block B_t can access all predicates from B_1 to B_{t-1}, as well as the input predicates. . 141

8.2 Grad-CAMs [139] and EigenCAMs [111] of the CNNs connected to an MLP (8.2a) and neural DNF-MT (8.2b) actor, at each unique step in the Door Corridor environment from Section 6.3.4. For each sub-figure, the left-to-right columns show the original inputs to the CNN, the Grad-CAMs, and the EigenCAMs in order. For each cell in the CAMs, the less greyed-out it is, the higher the activation it represents. 144

D.1 Relations between the sign-matching subsets of $\mathbf{x}_i$ and $\mathbf{x}_m$. 202

D.2 Relations between the sign-mismatching subsets of $\mathbf{x}_i$ and $\mathbf{x}_m$. 205

List of Tables

2.1 Confusion matrix for binary classification. The rows represent the actual condition (ground truth), and the columns represent the predicted condition. 14

3.1 Comparison of differentiable ILP approaches surveyed in this section. Bias column represents the type of search-space constraint. 34
3.2 Comparison of hybrid neuro-symbolic rule learning approaches surveyed in this section. 36
3.3 Comparison of end-to-end neuro-symbolic approaches for interpretable policy learning. Bias column represents the type of search-space constraint used. Obs. Type column represents the type of observations consumed by the agent. 39
3.4 Comparison of hybrid neuro-symbolic RL approaches that use classic ILP systems for structured knowledge induction. NeSy Task column represents the RL task addressed symbolically. 41

5.1 Statistics of different CUB datasets/subsets. Except for 'Original CUB-200-2011', all the other datasets/subsets are pre-processed. The number of samples is the sum of samples from all train, validation and test sets. For dataset names with a suffix '-k', the value of k indicates the number of classes retained in that dataset/subset. 69
5.2 Plain neural DNF models trained with binary cross entropy on different dataset/-subsets. All metrics are calculated by averaging across the validation set, except for the last column (the extracted rules' Jaccard score), which is computed on the test set. A plain neural DNF cannot learn under BCE with 200 classes, and thus the post-training process is not performed. 70
5.3 **Plain neural DNF** trained with **cross entropy** in different dataset/subsets. When using the Jaccard score as the metric during pruning with $\epsilon = 0.005$, all weights are pruned, resulting in no class being predicted. 71
5.4 **Neural DNF-EO model** trained with **cross-entropy loss** in different CUB subsets/dataset, at different stages of training and post-training pipeline. 71
5.5 Accuracy comparison among MLP, decision tree (no pruning) and trained neural DNF-EO model (no constraint layer C2, no post-training processing), across different CUB subsets/dataset. 72
5.6 Jaccard index score comparison across different CUB subsets/dataset, for FastLAS, trained neural DNF-EO model without the constraint layer C2 (no post-training processing), and the rules extracted from the post-training processed neural DNF-EO model without the constraint layer C2. FastLAS runs out of memory for CUB-200. . 73

5.7 Statistics of different TMC datasets/subsets. Except for 'Original TMC2007-500', all the other datasets/subsets are pre-processed. The number of samples is the sum of samples from all train, validation and test sets. For dataset names with a suffix '-k', the value of k indicates the number of labels retained in that dataset/subset. 75
5.8 Macro F1 of neural DNF models after training with BCE and at different stages of the post-training pipeline, MLPs trained with BCE, and decision trees without limiting depth, in different TMC subsets/dataset. 76
5.9 Macro F1 and the δ magnitude of a neural DNF model in TMC-22 dataset over 300 epochs of training. The drop in macro F1 occurs around when the δ magnitude is around 0.5. 76
5.10 MLP and neural DNF model's trainable parameters count, comparing to the number of non-leaf nodes in a decision tree without limiting depth, in different TMC subsets/dataset. A non-leaf node of a decision tree encodes a split condition that directs the decision path. 77
5.11 Statistics of the synthetic datasets. 78
5.12 Neural DNF-EO models' Jaccard index score on different synthetic multi-class datasets at different stages of the pipeline. 79
5.13 Neural DNF models' macro F1 score on different synthetic multi-label datasets at different stages of the pipeline. 79

6.1 Environment configurations for the three environments in the Switcheroo Corridor Environment Set, including SC's configuration. 92
6.2 Performance of the models in the Switcheroo Corridor MDP environment set, together with the ASP programs extracted from their corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean $\pm$ standard error. Argmax action selection is used for all models. 92
6.3 Performance of the models in the Switcheroo Corridor environment set, together with the ProbLog programs extracted from their corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean $\pm$ standard error. Different symbols after the actor's name indicate different action selection methods: † for ϵ-greedy sampling, and ‡ for actor's distribution sampling. 92
6.4 Blackjack: performance of MLP actors, neural DNF-MT actors, and the extracted ProbLog programs. Policy divergence measures the proportion of states where the argmax policy disagrees with the optimal policy from [150]. 93
6.5 Taxi: performance of Q-tables, MLP actors, neural DNF-MT actors, and the extracted ProbLog programs from the corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean $\pm$ standard error. Different symbols after the actor's name indicate different action selection methods: * for argmax action selection, and ‡ for actor's distribution sampling. 96
6.6 Door Corridor: performance of MLP actors, neural DNF-MT actors, and the extracted ASP programs from the corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean $\pm$ standard error. Argmax action selection is used for all models. 98

6.7 Optimal policy's actions for the Door Corridor (DC) environment and its variants, with blue text showing the difference in the variants. The difference in the optimal policy's actions is highlighted in blue. 99
6.8 Conjunctive and disjunctive nodes' tanh output when $x_2 = -1, x_7 = 1, x_9 = 1, x_{13} = -1$, calculated based on the formulation in Listing 6.7. Row 1 is the original output without applying thresholding, and row 2 is the output after thresholding on value 0. 101
6.9 Runtime comparison between different models across various environments. The overall runtime is the total time to run the model for the specified number of episodes. Models without * are run in a single environment in a loop for n times. Models with * are run in 8 parallel environments for a total of n episodes. 102

7.1 F1 scores (mean ± ste) for binary classification tasks. 'NDNF: ASP' is the logical interpretation of neural DNF-based models. F1 score is not measured for neural DNF-EO's ASP interpretation as there might be multiple classes included in the answer set and thus violating the exactly-one constraint of multi-class classification. 134
7.2 F1 scores (mean ± ste) for multi-class tasks. 'NDNF-MT: LI' is the logical interpretation of neural DNF-MT models. 134
7.3 Jaccard index scores (mean ± ste) for Neural DNF-EO models in multi-class tasks. 'NDNF-EO: ASP' is the logical interpretation of neural DNF-EO models. 134
7.4 F1 scores (mean ± ste) for multi-label tasks. 'NDNF: ASP' is the logical interpretation of neural DNF-based models. 134
7.5 F1 score drops after thresholding ($f1_t - f1_{thresh}$) and disentanglement ($f1_t - f1_{disent}$) discretisation. 135
7.6 Comparing compactness of neural DNF-based models' logic interpretations vs. decision trees', i.e. max. rule length/tree depth, avg. rule length/tree depth, and no. rules/branches. The lower the number the better the compactness. The cell marked with * refers to OSDT's performance. OSDT is only applicable when the input features are boolean and the target is a single class. 135

Acronyms

AI Artificial Intelligence

ASP Answer Set Programming

BCE Binary Cross Entropy

CE Cross Entropy

CNN Convolutional Neural Network

DNF Disjunctive Normal Form

ILP Inductive Logic Programming

LLM Large Language Model

MDP Markov Decision Process

MLP Multilayer Perceptron

NAF Negation as Failure

POMDP Partially Observable Markov Decision Process

PPO Proximal Policy Optimization

RL Reinforcement Learning

SVM Support Vector Machine

Notation

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

Functions

$\mathbb{1}(A)$	indicator function of event A, where $\mathbb{1}(A) = 1$ if event A occurs and 0 otherwise
$\mathbb{E}[X]$	expectation of a random variable X
$\arg\max_a f(a)$	value of a at which $f(a)$ is maximized
$\arg\min_a f(a)$	value of a at which $f(a)$ is minimized
$\text{sign}(x)$	sign function of x

Logic

$\bot$	truth value *false*
$\top$	truth value *true*

Neural Networks

$\mathbf{w}$	the weights of a node in a neural network
$\mathbf{x}$	the input to a node in a neural network
θ	the parameters of a neural network model
$\mathcal{L}(\theta)$	the loss function of a neural network with parameters θ

Semi-Symbolic Nodes and Neural DNF-based Models

$f_{\mathbf{w}}(\mathbf{x})$	the raw output of a semi-symbolic node with weights $\mathbf{w}$ and input $\mathbf{x}$
β	the logical bias of a semi-symbolic node, computed as $\delta\left(\max_i \lvert w_i\rvert - \sum_i \lvert w_i\rvert\right)$
δ	the parameter controlling conjunction ($\delta = 1$) or disjunction ($\delta = -1$) behaviour of a semi-symbolic node
$\mathbf{W}_{\text{c}}$	the weight matrix for the conjunctive layer of a neural DNF model
$\mathbf{W}_{\text{d}}$	the weight matrix for the disjunctive layer of a neural DNF model
$\mathcal{J}_{\mathbf{w}}$	the relevant set of indices of non-zero weights in $\mathbf{w}$, i.e. $\{j \mid w_j \neq 0\}$
$\mathcal{J}^+_{\mathbf{w}}$	the positive-sign relevant set of indices where $w_j > 0$
$\mathcal{J}^-_{\mathbf{w}}$	the negative-sign relevant set of indices where $w_j < 0$

b_{c}	the bivalent logical interpretation of a conjunctive node, where $b_{\mathrm{c}} = \top$ if the node output > 0 and $b_{\mathrm{c}} = \bot$ otherwise
b_{d}	the bivalent logical interpretation of a disjunctive node

Sets and Sequences

$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
$\lvert\mathcal{X}\rvert$	the number of elements in a finite set $\mathcal{X}$, or the cardinality of a finite set $\mathcal{X}$
$2^{\mathcal{X}}$	the power set of a finite set $\mathcal{X}$
$X[t]$	t-th element in a sequence X
$X[t{:}]$	subsequence of a sequence X starting from the t-th element

Chapter 1

Introduction

Deep learning has achieved remarkable success across diverse domains, demonstrating transformative impacts in healthcare [48, 104], autonomous systems [83], finance [42], natural language processing [40, 116], and computer vision [90, 66, 117]. These advances have been driven by the combination of computational power [153, 120] and data availability [149, 81], powering deep neural networks to learn complex patterns from large-scale data and even surpass human-level performance on specific tasks [145, 19]. However, despite their successes, neural networks are black-box models whose decision-making process remains largely inscrutable to human understanding, due to the high complexity from multiple compositions of non-linear transformations [133]. This opacity hinders these models' applicability in high-stakes domains such as medical diagnosis [70] and safety-critical systems [8], where transparency and accountability are paramount. While numerous post-hoc methods, such as LIME [130], SHAP [99], and saliency maps [139], have been developed for explaining model behaviours, these methods fundamentally provide *local* explanations that describe feature importance for individual predictions rather than offering a global, mechanism-level understanding of how the model reasons across the input space. Furthermore, research has revealed significant limitations: some saliency methods are independent of the model parameters and training data [2] and fail to provide sufficient information for explaining network accuracy and class relationships [168]; features identified as critical by methods like LIME and SHAP often have limited impact on actual network decisions [96]; and these interpretations can be fragile under small input shifts or perturbations [85, 61]. In contrast, models that are *inherently interpretable* are designed to provide transparent reasoning. They can offer more reliable and trustworthy explanations and are therefore more suitable for high-stakes applications [133].

Symbolic AI systems, which dominated the field before the deep learning era, offer a compelling foundation for interpretability: they operate on explicit rules and logical structures that are inherently human-readable, providing a more global explanation of the model's behaviour than attribution methods like LIME, SHAP or saliency maps. However, traditional symbolic approaches struggle with learning from raw data and lack the flexibility to handle noisy and real-world inputs. This motivates a natural question: can we combine the learning power of neural networks with the transparent reasoning of symbolic systems to achieve models that are both capable and interpretable?

Neuro-symbolic AI emerges as a promising answer to this question. It is a paradigm that integrates the pattern recognition capabilities of neural networks with the logical inference and explicit

knowledge representation of symbolic systems. This hybrid approach aims to overcome the lack of interpretability of standalone neural methods, while addressing the brittleness, scalability, and inability to learn from unstructured data of purely symbolic systems [134, 36, 20, 54]. Bridging the representation gap between the *sub-symbolic space* of real-valued vector activations or weights in neural networks and the *symbolic space* of discrete representations and reasoning lies at the core of neuro-symbolic approaches [134]. Existing approaches can be broadly grouped by how tightly they integrate neural and symbolic components.

At one end of the spectrum, *modular approaches* maintain neural and symbolic components as separate modules, interpreting neural components' output as symbols and feeding them to symbolic reasoners or learners. Approaches like DeepProbLog [101] and NeurASP [165] deploy symbolic reasoners and compute training signals for neural networks based on reasoning results and downstream labels, while NSIL [32] jointly learns a neural perception model and a symbolic rule-based model from raw data. The success of Large Language Models (LLMs) has inspired many works that connect LLMs with symbolic reasoners: many approaches use LLMs as translators to generate symbolic representations from unstructured inputs and then use symbolic solvers over such symbolic representations to solve downstream tasks [119, 75, 114], while others use LLMs to implement interpretation functions that generate truth valuations, directly embedding LLMs within symbolic reasoners' semantics [7]. These modular approaches benefit from leveraging mature neural and symbolic systems but face challenges in seamless end-to-end optimisation.

At the other end of the spectrum, *differentiable ILP approaches* embed symbolic knowledge and computation directly within neural architectures, enabling tighter integration and fully differentiable training. These approaches build upon the framework of inductive logic programming (ILP) [110, 92] and aim to learn interpretable rules through gradient-based optimisation, often relying on human-engineered rule templates [50, 140, 64] or mode biases [39] to restrict their search space. In traditional ILP, background knowledge refers to the set of known concepts available before learning (usually denoted B). In practice, however, rule templates and mode biases also form part of the prior knowledge of the task, as both require human engineering and tuning to ensure that a solution is included in the hypothesis space and that the search remains tractable for the learning system. This thesis uses the term background knowledge to refer to this overall prior knowledge. The ability to use such knowledge is an advantage inherited from ILP, allowing methods to leverage domain expertise and constrain the search to plausible hypotheses. However, this dependence can also limit structural flexibility and wider applicability [128]: if prior knowledge is unavailable, the hypothesis space may be too large; if the injected knowledge is incorrect, an optimal solution may be excluded entirely. Additionally, these methods typically require predicate-based inputs, or if inputs are unstructured, they necessitate either manual definition [39] or extraction using pre-trained components [84, 31]. Despite using neural networks as a core component, differentiable ILP methods fail to fully leverage the capabilities of these networks. Neural networks' approximation abilities extend naturally to unstructured data, and their expressivity allows them to cover large hypothesis spaces with minimal task-specific engineering: even small and shallow architectures demonstrate strong capabilities in learning complex boolean functions [113]. This limitation motivates the central research question of this thesis: how to develop a neuro-symbolic approach in an end-to-end differentiable model that fully exploits the approximation power and flexibility of neural networks while preserving interpretability?

This thesis addresses the question with the aim of developing a novel neuro-symbolic architecture

that achieves the following properties:

1. **Interpretability**: the model's behaviour can be described in a *logical form*.
2. **Adaptability**: the model leverages the expressivity of MLPs with *simple architectures* and can be easily set up for training with *different tasks and input formats*, *without* the need of encoding task-specific background knowledge into the model.
3. **End-to-end differentiability**: the model can be trained jointly with other neural components, enabling *fully end-to-end symbolic induction* without relying on manual feature engineering or pre-trained modules. In such a configuration, both the feature extractor and the neuro-symbolic learner are optimised simultaneously through gradient-based training, allowing the former to discover relevant high-level representations while the latter induces symbolic rules from these learned features.

The work presented in this thesis takes inspiration from the neural disjunctive normal form (DNF) model proposed in pix2rule [28]. With a design nearly identical to that of MLPs, a neural DNF model is an end-to-end differentiable model which, under appropriate conditions, can be translated to formal logic programs after training. Unlike previously mentioned neuro-symbolic approaches, the neural DNF model offers a broader space of possible rules and seamlessly integrates with other neural components. However, pix2rule evaluates the neural DNF model only on synthetically generated image classification tasks with a single output class, leaving open the question of how well the model generalises to real-world datasets and other task settings. Additionally, the conditions under which the translation from neural weights to logical rules preserves semantic equivalence are not formally established.

This thesis proposes a family of neural DNF-based models, based on the notion of neural DNF model, with wide applicability to various tasks and input formats, as well as theoretical guarantees for logical interpretability, thus better fulfilling the three desired properties of interpretability, adaptability and end-to-end differentiability defined above. Therefore, the thesis statement is as follows:

> **Thesis Statement.** *The family of neural DNF-based models proposed in this thesis can perform differentiable rule learning for classification and reinforcement learning tasks without prior knowledge, offering logical interpretability, intervention capabilities, and end-to-end integration with other neural components to support predicate invention.*

The following Section 1.1 summarises the contributions of this thesis in three parts: theoretical foundations, applications, and model discretisation. Section 1.2 lists out the publications this thesis builds upon and Section 1.3 outlines the structure of this thesis.

1.1 Contributions

1.1.1 Theoretical Foundations

The core computational units of the proposed neural DNF-based approaches are semi-symbolic nodes, which approximate logical conjunctions or disjunctions.[1]The main theoretical contribution of

[1]The essential background on semi-symbolic nodes is provided in Section 2.3.1.

this thesis, presented in Chapter 4, is a formal framework that bridges the neural and bivalent logic forms of these semi-symbolic nodes. Specifically, this thesis:

- formalises the behaviour and interpretation of semi-symbolic nodes;
- mathematically proves that the translation between neural and bivalent logic forms is bidirectional and preserves truth value equivalence.

These results establish the theoretical guarantees absent in the prior work pix2rule [28]. The neural-to-bivalent-logic translation enables interpretability by converting learned weights into logical rules, as demonstrated throughout this thesis. Conversely, the bivalent-logic-to-neural translation enables intervention on the neural model's behaviour by modifying learned rules and converting them back to neural form, as demonstrated in Section 6.3.5.

1.1.2 Neural DNF-based Models for Classification and Reinforcement Learning

A family of neural DNF-based models is proposed in this thesis, which perform differentiable rule learning for classification and reinforcement learning tasks. Chapter 5 evaluates the neural DNF model on non-synthetic real-world classification tasks. It discusses the difficulty of enforcing a *deterministically mutually-exclusive constraint* with the neural DNF model while supporting bivalent logic interpretation for multi-class classification tasks. Thus, the neural DNF-EO model is proposed to enforce such a constraint in a neural DNF-based architecture. To address the challenge of interpretable and editable policy learning in reinforcement learning, the DNF-MT model is proposed in Chapter 6, which outputs probability distributed predictions with a *stochastically mutually-exclusive constraint*. The trained model can be interpreted as a probabilistic logic program and under certain conditions as a bivalent logic program. Chapter 6 reports the experiments where the neural DNF-MT model is used to learn interpretable and editable policies for reinforcement learning tasks, and the experiments in Chapter 7 show that the neural DNF-MT model is more stable than the neural DNF-EO model in classification tasks. In both classification and RL settings, the neural DNF-based models achieve performance comparable to competing approaches while offering concise and modifiable logical representations.

1.1.3 Model Discretisation: from Continuous Space to Discrete Space

A central challenge in neuro-symbolic AI is the translation from continuous neural representations to discrete symbolic forms [134]. For a neural DNF-based model, this difficulty occurs during the process of translating the model's learned hypothesis encoded in real-valued weights to a set of discrete values that can be interpreted as logical rules. This continuous-to-discrete transformation requires discretely approximating a soft-valued model, which introduces discontinuities into the neural representation and may cause catastrophic performance degradation. Figure 1.1 illustrates this problem with an example of a learned continuous hypothesis and its counterpart after discretisation. The discretisation process pushes the peaks of the original hypothesis tighter around the discrete targets, but this transformation also changes the behaviour of the function in unwanted ways. In particular, discrete targets that were originally covered by the original hypothesis may be missed after discretisation, leading to performance degradation. This problem is persistent across various experiments in

Chapter 5 and Chapter 6. Chapter 7 identifies the weights entanglement problem in trained neural DNF-based models where soft-valued weights of a single node encode multiple discrete rules rather than learning to represent a single discrete rule. A disentanglement method is thus proposed to split such an entangled node into multiple independent nodes that jointly encode the same behaviour as the original node. This method significantly reduces the performance degradation when translating trained neural DNF-based models into logical forms.

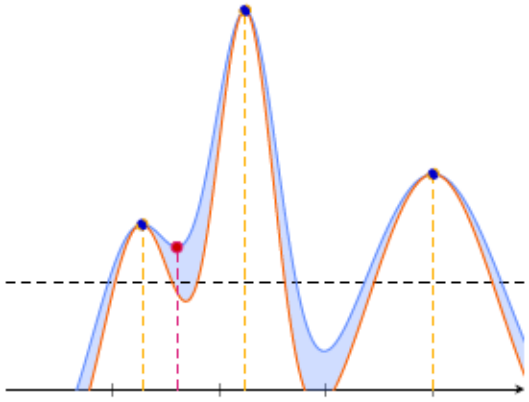


Figure 1.1: An illustration of an original learned continuous hypothesis in blue and a hypothesis after some degree of discretisation in orange. The discrete targets that the hypothesis is expected to cover are marked with dots (with yellow or red dashed lines connected to them). The discretisation process shrinks the area of coverage by pushing the peaks tighter around the discrete targets, and the area of the difference between the original and hypothesis after discretisation is shaded in blue. However, it also changes the behaviour of the function. If the value is interpreted as true when the hypothesis is above the black dashed line, then there is a discrete target marked in red that was covered by the original hypothesis in blue but is now missed by the new hypothesis in orange.

1.2 Publications

This thesis contains revised and expanded content from the following publications, listed in chronological order:

- Kexin Gu Baugh, Nuri Cingillioglu, and Alessandra Russo. Neuro-symbolic Rule Learning in Real-world Classification Tasks. In Andreas Martin, Hans-Georg Fill, Aurona Gerber, Knut Hinkelmann, Doug Lenat, Reinhard Stolle, and Frank van Harmelen, editors, *Proceedings of the AAAI 2023 Spring Symposium on Challenges Requiring the Combination of Machine Learning and Knowledge Engineering (AAAI-MAKE 2023)*, volume Vol-3433. CEUR Workshop Proceedings, 2023. The material is used in Chapter 5.

- Kexin Gu Baugh, Luke Dickens, and Alessandra Russo. Neural DNF-MT: A Neuro-symbolic Approach for Learning Interpretable and Editable Policies. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '2025, pages 252-260, Richland, SC, 2025. International Foundation for Autonomous Agents and Multiagent Systems. The material is used in Chapter 4 and Chapter 6.

- Kexin Gu Baugh, Vincent Perreault, Matthew Baugh, Luke Dickens, Katsumi Inoue and Alessandra Russo. Disentangling Neural Disjunctive Normal Form Models. In Leilani H. Gilpin, Eleonora Giunchiglia, Pascal Hitzler, and Emile van Krieken, editors, *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning (NeSy 2025)*, volume 284 of Proceedings of Machine Learning Research, pages 463-493. PMLR, 08-10 Sep 2025. The material is used in Chapter 4 and Chapter 7.

1.3 Thesis Structure

The remainder of this thesis is organised into seven chapters. The first two chapters establish the necessary foundations: Chapter 2 introduces the relevant background in symbolic and neural methods, while Chapter 3 situates this thesis within the broader landscape of neuro-symbolic learning and neuro-symbolic reinforcement learning.

The technical contributions begin with Chapter 4, which develops the theoretical underpinnings by formalising the semi-symbolic nodes and proving the bidirectional translation between neural DNF-based models' neural and bivalent logic forms. The subsequent two chapters demonstrate practical applications: Chapter 5 proposes the neural DNF-EO model for multi-class classification with deterministic mutual exclusivity, while Chapter 6 introduces the neural DNF-MT model for learning interpretable policies in reinforcement learning with probabilistic logical interpretation. Chapter 7 then addresses the challenge of translating continuous representations to discrete representations, proposing a disentanglement method to better preserve model behaviour. The thesis concludes with Chapter 8, summarising the contributions and outlining directions for future work.

Chapter 2

Background

This chapter provides the background knowledge necessary to understand the key advances presented in this thesis. As this work sits at the intersection of symbolic reasoning and deep learning, it draws upon concepts from both fields. The chapter begins with Answer Set Programming (Section 2.1), which introduces the logical formalism used to represent and interpret the rules learned by the neural DNF-based models. It then covers core concepts of deep learning (Section 2.2), establishing the neural network foundations upon which the models are built. Following this, the neural DNF model proposed in pix2rule [28] is presented (Section 2.3), as it serves as the starting point for the family of neural DNF-based models developed in this thesis. Finally, the chapter introduces reinforcement learning (Section 2.4), which provides the framework for one of the key application domains explored in later chapters. Together, these foundations equip the reader with the necessary tools to follow the technical contributions that follow.

2.1 Answer Set Programming

Answer Set Programming (ASP) [95, 46] is a declarative programming paradigm that represents knowledge in the form of logical rules based on the stable model semantics [60]. The principle of ASP is to formulate a problem as a non-monotonic logic program and generate a solution to the problem by computing the stable model of the program. Stable models of ASP programs are also called answer sets. This section first introduces the syntax of logic programs before proceeding to their semantics.

Syntax. The basic syntactic elements of a logic program are built hierarchically. A *term* can take three forms: it may be a constant (e.g., `a`), a variable (e.g., `X`), or an n-ary function symbol applied to n terms (e.g., `f(a,X)`). When a term contains no variables, it is said to be *ground*. An *atom* consists of a predicate symbol of arity n together with n terms as arguments (e.g., `P(X,a)`); an atom is ground when all of its argument terms are ground. A *literal* refers to either an atom or its negation. ASP distinguishes between classical negation ($\neg$) negation and negation as failure (NAF, written `not`). NAF underpins non-monotonic reasoning and ASP's stable model semantics: under the closed-world assumption, if an atom cannot be derived from the program, it is deemed to be false.

Definition 2.1.1 (Clause). *A* clause *(or a* rule*) r has the following structure:*

$$H \leftarrow B_1, B_2, \ldots, B_n, \texttt{not } C_1, \texttt{not } C_2, \ldots, \texttt{not } C_m$$

where $n, m \geq 0$. The atom H is the head *of the clause, written head(r). The conjunction $B_1, \ldots, B_n$, $\texttt{not } C_1, \ldots, \texttt{not } C_m$ constitutes the* body*, written body(r). The set of positive body atoms is referred to as the positive body, denoted $\text{body}^+(r) = \{B_1, \ldots, B_n\}$, and the set of atoms appearing under NAF is referred to as the negative body, denoted $\text{body}^-(r) = \{C_1, \ldots, C_m\}$. A clause is* ground *when all its atoms are ground. A clause is* definite *if it has an empty negative body.*

A clause or a rule can be understood as a conditional statement: if all body literals are satisfied, then the head follows. A clause such as $H \leftarrow$ with an empty body is called a *fact*. A *logic program* P is a finite set of clauses.

Logic formulae can be transformed into standardised representations known as *normal forms*. Two widely used normal forms are Conjunctive Normal Form and Disjunctive Normal Form.

Definition 2.1.2 (Conjunctive Normal Form). *A formula is in* Conjunctive Normal Form *(CNF) if it is a conjunction of one or more disjunctions of literals. A CNF formula has the following structure:*

$$\bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_i} l_{ij} \right)$$

where each l_{ij} is a literal.

Definition 2.1.3 (Disjunctive Normal Form). *A formula is in* Disjunctive Normal Form *(DNF) if it is a disjunction of one or more conjunctions of literals. A DNF formula has the following structure:*

$$\bigvee_{i=1}^{m} \left(\bigwedge_{j=1}^{n_i} l_{ij} \right)$$

where each l_{ij} is a literal.

For example, $(a \vee \neg b) \wedge (c \vee d)$ is a CNF formula with two disjunctions, and $(a \wedge \neg b) \vee (c \wedge d)$ is a DNF formula with two conjunctions. A DNF formula can be represented as a set of clauses, where each clause has a conjunction of the DNF formula as its body and the head in all clauses is an auxiliary literal to represent the whole DNF formula. For example, the DNF formula $(a \wedge \neg b) \vee (c \wedge d)$ can be represented as the set of clauses $\{h \leftarrow a, \neg b.\ \ h \leftarrow c, d.\}$, where h is an auxiliary literal to represent the original DNF formula. A neural DNF-based model is designed to learn formulae in DNF form, where the conjunctive layer computes individual conjuncts used in the conjunctions and the disjunctive layer combines the conjunctions into a disjunction. This allows the flexibility of learning different conditions for the same target, i.e., a set of rules with the same head (target) but different bodies (conditions).

Semantics.

Definition 2.1.4 (Herbrand Universe). *Given a logic program P, the* Herbrand universe *$\mathcal{HU}_P$ comprises all ground terms constructible using the constants and function symbols that appear in P.*

Note that a Herbrand universe can be infinite with the recursive application of function symbols. In the absence of function symbols, the Herbrand universe reduces to the set of constants occurring in P.

Definition 2.1.5 (Herbrand Base). *The* Herbrand base $\mathcal{HB}_P$ *of a logic program P is the set of all ground atoms that can be formed by applying the predicates in P to terms from the Herbrand universe $\mathcal{HU}_P$.*

Definition 2.1.6 (Herbrand Interpretation). *An* Herbrand interpretation I *of a logic program P is a subset of the Herbrand base $\mathcal{HB}_P$ and is a truth assignment to the atoms in $\mathcal{HB}_P$. The atoms belonging to I are deemed true, while those outside I are deemed false.*

Definition 2.1.7 (Grounding). *The* grounding *of a logic program P, written ground(P), is the program obtained by replacing each clause in P with all of its ground instances. A ground instance of a clause is obtained by applying a substitution that maps every variable in the clause to a ground term from $\mathcal{HU}_P$. The resulting program contains only ground clauses.*

Once a first-order logic program has been grounded, each ground atom (e.g., $\mathtt{P(a,b)}$) can be treated as a propositional term, and the grounded clauses become propositional clauses. In this sense, a grounded first-order program $ground(P)$ is equivalent to a propositional program: it possesses the same logical structure and can be evaluated using propositional reasoning techniques. This equivalence is relevant to the present thesis, which operates primarily within propositional logic.

Modern ASP solvers such as Clingo [126] first ground the program and then compute the stable models of the grounded program. To compute the stable models of a grounded program, ASP solvers construct the reduct of the program.

Definition 2.1.8 (Reduct). *Let P be a ground logic program and I an interpretation. The* reduct *of P with respect to I, denoted P^I, is constructed through the following two steps:*

1. *Remove any rule r for which some atom in its negative body also belongs to I (i.e. $a \in body^-(r)$ and $a \in I$).*
2. *Remove any negative body literals from the bodies of the remaining rules.*

The resulting reduct can be expressed as:

$$P^I = \left\{ head(r) \leftarrow body^+(r) \mid r \in P \text{ and } body^-(r) \cap I = \emptyset \right\}$$

P^I is now a definite ground program. An interpretation I is a *stable model* (or *answer set*) of P if I is the unique minimal Herbrand model of the reduct P^I.

Modern ASP systems also include powerful extensions to the basic ASP language, such as choice rules, constraints, aggregations and more. They allow for more expressive and efficient problem formulations. This thesis focuses solely on representing knowledge in the form of rules (or clauses) using the basic ASP syntax, as shown in Definition 2.1.1, and does not cover these language extensions.

2.2 Deep Learning

Deep learning [161] has revolutionised Artificial Intelligence (AI) by enabling machines to learn hierarchical representations from data through multi-layered neural networks. From AlexNet [90] in 2012

to GPT-5 [116] in 2025, deep learning has achieved remarkable success in various domains including computer vision [66, 117], natural language processing [40, 116], and reinforcement learning [19, 145]. The core of deep learning is the neural network architecture, which consists of interconnected layers of artificial neurons that transform input data through a series of non-linear operations. This section introduces the foundational concepts and methodologies underpinning this thesis, encompassing neural network architectures, training procedures, regularisation techniques, and classification tasks.

2.2.1 Neural Networks

Most modern deep learning models are built by stacking multiple computational modules called layers. A layer is a functional block of mathematical operations that transforms a stack of input values into an associated stack of output values that are more abstract representations of the input. Throughout the years, the types of layers used in sophisticated deep learning models have expanded, including linear layers, convolutional layers, recurrent layers, and attention layers. The most basic and common type of layer used in deep learning is the linear layer, which computes:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

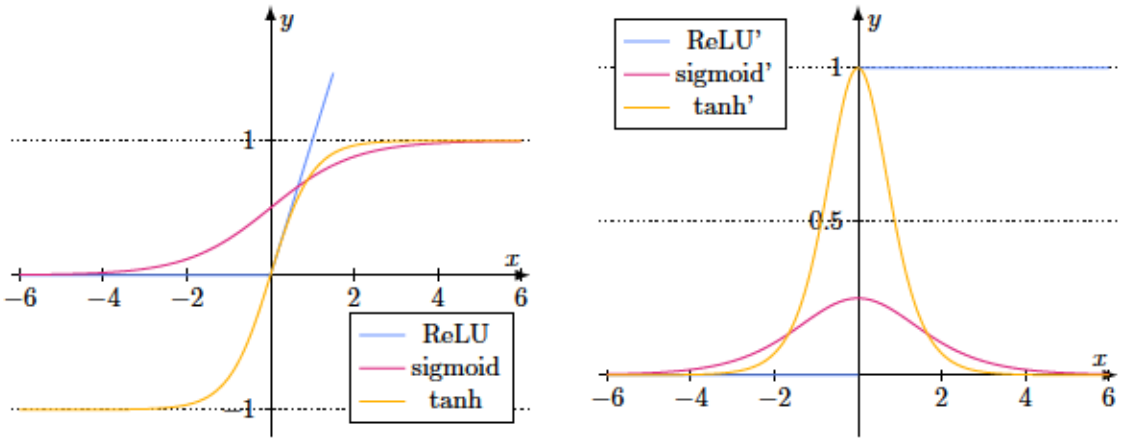
where $\mathbf{x} \in \mathbb{R}^N$ is the input to the layer, $\mathbf{W} \in \mathbb{R}^{M \times N}$ is the learnable weight matrix, and $\mathbf{b} \in \mathbb{R}^M$ is the learnable bias vector. Strictly speaking, this is an affine transformation due to the bias term, but it is conventionally referred to as a linear layer. A single linear layer without activation can only represent linearly separable functions. By introducing non-linear activation functions and stacking multiple layers, neural networks become capable of learning complex, non-linear patterns. Common non-linear activation functions include the sigmoid function (Equation (2.1)) that maps inputs to the range $(0, 1)$, and tanh (Equation (2.2)) that produces outputs in $(-1, 1)$. However, these two functions suffer from the vanishing gradient problem (Figure 2.1b), where gradients become exponentially small during back-propagation through deep networks, hindering the learning [18, 65]. The Rectified Linear Unit (ReLU) (Equation (2.3)) is more widely used than sigmoid and tanh due to its computational efficiency and ability to mitigate the vanishing gradient problem. ReLU allows gradients to flow unchanged for positive inputs whilst completely blocking negative values, which enables faster training and better performance in deep architectures. However, ReLU suffers from the "dying ReLU" problem, where neurons can become permanently inactive. Variants such as Leaky ReLU, which applies a small slope to negative inputs, and Parametric ReLU (PReLU), where the slope is learned during training, are used to address this problem. Most neural DNF-based models in this thesis use the tanh activation function for interpretability requirements, since the tanh function outputs fixed-range values that can be easily used for logical interpretation (more details in Section 2.3.1) compared to the unbounded range of ReLU-based activations. And compared to the sigmoid function, the tanh function is zero-centred and has a bigger range of gradients, making it more resilient to the vanishing gradient problem. However, the use of tanh can still lead to vanishing

gradient problems practically if the network is too deep.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.3}$$



(a) Comparison of the activation functions.

(b) Comparison of the derivatives of the activation functions.

Figure 2.1: Comparison of the activation functions and their derivatives.

As a simple yet powerful baseline model, Multi-Layer Perceptron (MLP) consists of multiple linear layers and activation functions. The depth of the MLP refers to the number of layers in the network, and the width of the MLP refers to the number of neurons in each layer. The deeper and the wider the MLP, the more learnable parameters it has and the more expressive the model becomes. An MLP with just one hidden layer and non-polynomial activation function can approximate any continuous function to arbitrary accuracy given a sufficient number of hidden units, a property known as the universal approximation theorem [93].

2.2.2 Optimisation

Training a neural network involves optimising its parameters θ via minimising a loss function that measures the discrepancy between predictions and ground truth labels. A pointwise loss function $l(\hat{y}, y)$ takes in predictions $\hat{y}$ computed by the model and ground truth label y and outputs a scalar loss value. The population loss $\mathcal{L}(\theta)$ is defined as the expected pointwise loss over the data distribution $\mathcal{D}$:

$$\mathcal{L}(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\left[l(f_\theta(\mathbf{x}), y)\right]$$

where f_θ denotes the model with parameters θ. And $\tilde{\mathcal{L}}(D;\theta)$ is the empirical loss computed on a dataset $\tilde{\mathcal{D}}$ with parameters θ:

$$\tilde{\mathcal{L}}(\tilde{\mathcal{D}};\theta) = \frac{1}{|\tilde{\mathcal{D}}|}\sum_{i=1}^{|\tilde{\mathcal{D}}|} l(f_\theta(\mathbf{x}_i), y_i)$$

where $|\tilde{\mathcal{D}}|$ is the number of samples in the dataset. Below are some common pointwise loss functions.

- Squared Error (SE) is commonly used in regression tasks where the output is continuous. It is defined as $l(\hat{y}, y) = (\hat{y} - y)^2$.
- Binary Cross Entropy (BCE) works on binary decisions by penalising mismatches between predictions and ground truth. It is defined as $l(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$.
- Categorical Cross Entropy (CE) extends BCE to multi-class classification tasks, where $\hat{\mathbf{y}}$ and $\mathbf{y}$ associated with a sample $\mathbf{x}$ are interpreted as probability distributions over C classes. It is defined as:

$$l(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{j=1}^{C} y_j \log \hat{y}_j.$$

The loss function optimisation is typically done via gradient-based methods that iteratively update parameters in the direction of steepest descent. Stochastic Gradient Descent (SGD) is the most fundamental optimisation algorithm, which updates parameters using gradients computed on small random subsets (mini-batches) B_t of the training data:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \tilde{\mathcal{L}}(B_t; \theta_t)$$

where η is the learning rate controlling the step size. Whilst SGD is simple and computationally efficient, its performance is sensitive to the choice of learning rate, and it can struggle with ill-conditioned loss surfaces. Adaptive optimisation algorithms address these limitations by automatically adjusting learning rates for individual parameters. Adam (Adaptive Moment Estimation) [86] is the most popular adaptive optimiser, which maintains exponentially decaying averages of past gradients (first moment) and squared gradients (second moment) to compute adaptive learning rates. The Adam update rule is given by:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta \tilde{\mathcal{L}}(B_t; \theta_t) \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_\theta \tilde{\mathcal{L}}(B_t; \theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \end{aligned}$$

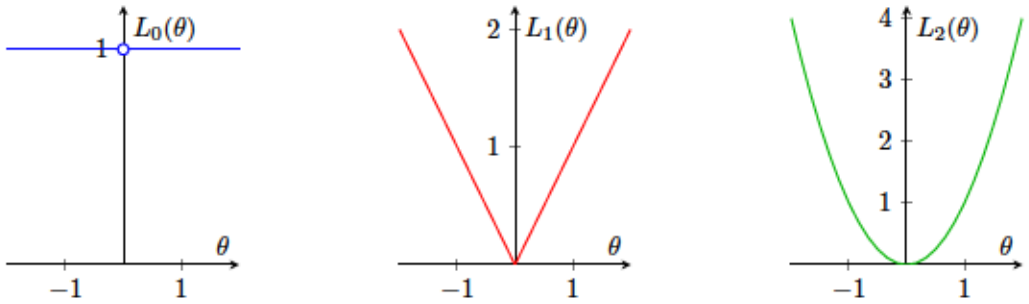
where β_1 and β_2 are decay rates for the moment estimates, and ϵ is a small constant for numerical stability. Adam combines the benefits of momentum, which accelerates convergence in relevant directions, with adaptive learning rates that handle sparse gradients effectively. The experiments in this thesis utilise Adam for optimisation.

2.2.3 Regularisation

Regularisation techniques are crucial for preventing overfitting, where a model performs well on training data but fails to generalise to unseen examples. These methods can be seen as additional constraints to the training process that encourage simpler models or more robust learned representations. L_p regularisations are based on the p-norm of the parameter vector. These L_p norm penalties are added with a hyperparameter λ as a 'soft constraint' to the base loss function $\mathcal{L}$, resulting the

overall objective function to be $\mathcal{L}(\theta) + \lambda L_p(\theta)$. The regularisation functions are defined as follows:

$$L_0(\theta) = \|\theta\|_0 = \sum_i \mathbb{I}(|\theta_i| > 0)$$
$$L_1(\theta) = \|\theta\|_1 = \sum_i |\theta_i|$$
$$L_2(\theta) = \|\theta\|_2^2 = \sum_i \theta_i^2$$



(a) L_0 regularisation. The penalty is constant at 1 for non-zero parameters and 0 for zero parameters.

(b) L_1 regularisation, using the absolute value to promote sparsity.

(c) L_2 regularisation, using the squared value to encourage even weight distribution.

Figure 2.2: Comparison of L_p regularisation functions.

L_1 regularisation promotes sparsity by theoretically driving many weights exactly to zero, effectively performing feature selection. L_2 regularisation penalises the squared magnitude of weights and encourages the network to distribute weights more evenly and prevents individual weights from becoming excessively large. In contrast to L_1 and L_2, L_0 regularisation directly penalises the number of non-zero parameters, inducing no shrinkage on the values of the parameters and thus yielding the sparsest solutions. But it is computationally intractable for gradient-based optimisation due to its non-differentiability and combinatorial nature. Approaches such as [98, 135] have been proposed to approximate L_0 regularisation for gradient-based optimisation.

2.2.4 Classification Tasks in Deep Learning

Classification is a supervised learning problem in deep learning where the objective is to assign input data to predefined categories or labels. Depending on the nature and constraints of the problem, classification tasks can be categorised into three main types: binary, multi-class, and multi-label classification. Each type has distinct characteristics, employs specific loss functions for training, and requires appropriate evaluation metrics.

Binary Classification

Binary classification involves predicting one of two possible classes for each input instance. The task requires assigning each sample to exactly one of the two classes, either a positive class (typically labelled 1) or a negative class (typically labelled 0). Mathematically, for an input $\mathbf{x}$, the model learns a function $f : \mathcal{X} \to \{0, 1\}$ that maps inputs to one of the two classes. The standard loss function

for binary classification is the binary cross-entropy (BCE) loss. This loss function penalises confident incorrect predictions more severely than uncertain ones, as the logarithm approaches negative infinity when the predicted probability diverges from the true label. Binary classification models are evaluated using several metrics derived from the confusion matrix (Table 2.1), which tallies true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN):

- **Accuracy** measures the proportion of correct predictions:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

  Whilst intuitive, accuracy can be misleading for imbalanced datasets where one class significantly outnumbers the other.

- **Precision** quantifies the proportion of positive predictions that are correct:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

- **Recall** (also called sensitivity or true positive rate) measures the proportion of actual positives correctly identified:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

- **F1 Score** provides a harmonic mean of precision and recall, offering a balanced metric particularly useful for imbalanced datasets:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}.$$

  The F1 score ranges from 0 to 1, with higher values indicating better performance. It penalises extreme imbalances between precision and recall, making it preferable to accuracy when both false positives and false negatives carry significant cost.

		Predicted condition	
		Positive	Negative
Actual condition	Positive	True positive	False negative
	Negative	False positive	True negative

Table 2.1: Confusion matrix for binary classification. The rows represent the actual condition (ground truth), and the columns represent the predicted condition.

Multi-class Classification

Multi-class classification extends binary classification to problems with $C > 2$ classes. Each instance must be assigned to *exactly one* class from the set $\{1, 2, \ldots, C\}$. The model learns a function $f : \mathcal{X} \to \{1, 2, \ldots, C\}$ that maps inputs to a single class label. The standard loss function for multi-class classification is the categorical cross-entropy (CE) loss, and similar metrics extended to handle multiple classes are computed analogously to binary classification. Accuracy remains the proportion

of correctly classified instances:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}.$$

Precision, Recall, and F1 Score can be computed per class or aggregated across classes using different averaging strategies:

- **Macro-averaging**: Calculate the metric for each class independently and take the unweighted mean, treating all classes equally regardless of their frequency. For class i, precision and recall are defined as:

  $$\text{Precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i}, \quad \text{Recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i}.$$

  Here, TP_i, FP_i, and FN_i denote the true positives, false positives, and false negatives for class i, respectively, when treating class i as the positive class in a one-vs-rest manner. The per-class F1 score, F1_i, is computed analogously to the binary case. The macro-averaged F1 score is then:

  $$\text{Macro F1} = \frac{1}{C}\sum_{i=1}^{C} \text{F1}_i.$$

  Since each class contributes equally to the final score, macro-averaging is particularly sensitive to performance on minority classes. Poor performance on a rare class will significantly reduce the macro score, even if the model performs well on frequent classes. This makes macro-averaging suitable when all classes are equally important, regardless of their prevalence in the dataset.

- **Micro-averaging**: Compute the global average by summing the counts for every class to get the total true positives, true negatives, false positives, and false negatives, and then compute the precision, recall, and F1 score:

  $$\begin{aligned}
  \text{Micro Precision} &= \frac{\text{Total TP}}{\text{Total TP} + \text{Total FP}}, \\
  \text{Micro Recall} &= \frac{\text{Total TP}}{\text{Total TP} + \text{Total FN}}, \\
  \text{Micro F1} &= \frac{2 \cdot \text{Total TP}}{2 \cdot \text{Total TP} + \text{Total FP} + \text{Total FN}}.
  \end{aligned}$$

  Here, $\text{Total TP} = \sum_{i=1}^{C} \text{TP}_i$, $\text{Total FP} = \sum_{i=1}^{C} \text{FP}_i$, and $\text{Total FN} = \sum_{i=1}^{C} \text{FN}_i$ are the sums of per-class counts. In contrast to macro-averaging, micro-averaging is dominated by performance on majority classes, as frequent classes contribute more to the aggregate counts. A model that performs well on common classes but poorly on rare ones may still achieve a high micro score. This approach is appropriate when overall prediction accuracy matters more than balanced performance across all classes.

- **Weighted-averaging**: Compute the metric for each class and average them weighted by the number of true instances for each class. This approach provides a more balanced summary metric of the model's performance across all classes. It accounts for class imbalance by giving more weight to frequent classes, whilst still computing per-class metrics that can reveal poor performance on specific classes.

(Binary) Multi-label Classification

In multi-label classification, instances may be associated with *multiple labels simultaneously*. This thesis considers the binary variant, where each label indicates the presence or absence of an attribute. Unlike multi-class classification's mutual exclusivity constraint, multi-label classification allows any subset of labels from the label set $\mathcal{L} = \{1, 2, \ldots, L\}$ to be true for a given instance. The model learns a function $f : \mathcal{X} \to \{0,1\}^L$ that predicts a binary vector indicating the presence or absence of each label. This formulation is particularly relevant for applications such as tagging and categorisation, where objects naturally belong to multiple categories. Multi-label classification typically employs BCE loss applied independently to each label. A sigmoid activation function is applied independently to each label's logit to produce probabilities, allowing each label to be predicted independently without the normalisation constraint imposed by softmax in multi-class classification. Multi-label classification requires metrics that account for partial correctness, as predictions may be partially accurate. For a dataset with N samples and L labels, the metrics are defined as follows:

- **Subset Accuracy (exact match ratio)** is a strict metric requiring the entire predicted label set to exactly match the true label set:

$$\text{Subset Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(\mathbf{y}_i = \hat{\mathbf{y}}_i).$$

  Here, $\mathbb{1}(\cdot)$ is the indicator function: $\mathbb{1}(\alpha) = 1$ if condition α is true and 0 otherwise. $\mathbf{y}_i \in \{0,1\}^L$ and $\hat{\mathbf{y}}_i \in \{0,1\}^L$ are the true and predicted label vectors for the i-th sample, respectively.

- **Hamming Loss** measures the fraction of incorrectly predicted labels, averaged over all samples and labels:

$$\text{Hamming Loss} = \frac{1}{NL}\sum_{i=1}^{N}\sum_{j=1}^{L} \mathbb{1}(y_{i,j} \neq \hat{y}_{i,j}).$$

  Here, $y_{i,j}$ and $\hat{y}_{i,j}$ are the j-th element of the true and predicted label vectors for the i-th sample, respectively. Lower values indicate better performance, with 0 representing perfect classification. Hamming loss is more forgiving than exact match accuracy, as it penalises individual label errors rather than requiring perfect label set prediction.

- **Precision, Recall, and F1 Score** can be extended to multi-label settings using various averaging schemes, similar to multi-class classification.

2.3 Semi-Symbolic Node and Neural DNF Model

This section first summarises the formulation and the training of the semi-symbolic node and the neural DNF model as introduced in pix2rule [28], which provide the basic ingredients of the neural DNF-based models proposed in this thesis. This section also recaps the post-training process that translates a trained neural DNF model to an ASP program, as originally presented in pix2rule [28]. Subsequent chapters of this thesis adapt and refine these processes for the neural DNF-based models proposed in this thesis.

2.3.1 Model Formulation

In pix2rule [28], a semi-symbolic node with trainable weights w_i, $i = 1, \ldots, N$ and parameter δ, with input $\mathbf{x} \in [-1, 1]^N$, is given by:

$$f_{\mathbf{w}}(\mathbf{x}) = \left(\sum_{i=1}^{N} w_i x_i + \beta \right) \tag{2.4}$$

$$\beta = \delta \left(\max_{i=1}^{N} |w_i| - \sum_{i=1}^{N} |w_i| \right) \tag{2.5}$$

$$\hat{y} = \tanh\left(f_{\mathbf{w}}(\mathbf{x})\right) \tag{2.6}$$

Here the N (semi-symbolic) inputs to the node are constrained such that $x_i \in [-1, 1]$, where the extreme value 1 (resp. -1) is interpreted as the associated term i taking the logical value $\top$ (resp. $\bot$) with other values representing intermediate strengths of belief (a form of fuzzy logic or generalised belief). The node activation $\hat{y} \in (-1, 1)$ is interpreted similarly, but cannot take specific values 1 or -1. The node's characteristics are controlled by a hyperparameter δ, which induces behaviour analogous to a logical conjunction (resp. disjunction) when $\delta = 1$ (resp. -1) precisely. The neural DNF model is a feedforward neural network with a layer of conjunctive nodes followed by a layer of disjunctive nodes to mimic a DNF formula, as shown in Figure 2.3.

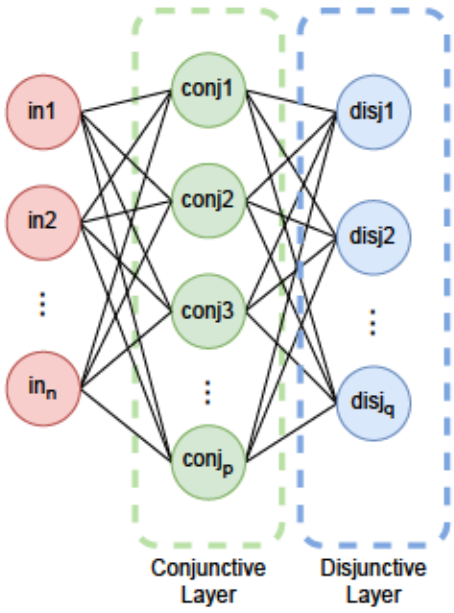


Figure 2.3: A neural DNF model with a conjunctive layer followed by a disjunctive layer. Conjunctive nodes in the conjunctive layer have their δ values greater than 0, while disjunctive nodes in the disjunctive layer have their δ values less than 0.

2.3.2 Training

The training of the neural DNF model remains mostly the same as the training of an MLP, relying on backpropagation and gradient descent to update the weights of the semi-symbolic nodes. The key difference is that the logical bias β is not a trainable parameter but calculated as Equation (2.5). δ in the bias term is not a trainable parameter but its magnitude is controlled by a scheduler to increase throughout training such that $|\delta| = 1$ at convergence. The reason for scheduling δ is to avoid the model failing to learn any rules if the logical bias is at full strength at the beginning of training. In pix2rule, after some delay $t_{\text{delay}} \in \mathbb{N}$, $|\delta|$ is updated every $t_{\text{steps}} \in \mathbb{N}_{>0}$ steps, increasing from a small initial magnitude $\delta_0 \in (0, 1)$ to the magnitude of 1 in an exponential growth with rate $r > 1$:

$$|\delta_{t+1}| = \begin{cases} \delta_0 & \text{if } t < t_{\text{delay}} \\ \min\left(\delta_0 \cdot r^{\lfloor \frac{t - t_{\text{delay}}}{t_{\text{steps}}} \rfloor + 1}, 1\right) & \text{if } t \geq t_{\text{delay}} \end{cases} \tag{2.7}$$

Figure 2.4 shows an example of the delta scheduling proposed in pix2rule. By adjusting δ_0, r, t_{delay}, and t_{steps}, the neural DNF model can steadily adjust the strength of the logical bias during training and learn in a stable manner. Most experiments conducted in this thesis use this basic delayed exponential scheduling, as defined in Equation (2.7), to update the magnitude of δ throughout training while maintaining learning stability. However, some experiments in Section 7.5 show instabilities in training when learning from complex datasets, and Section 7.4.2 explores alternative delta scheduling methods including adaptive scheduling that monitors the model's performance during training.

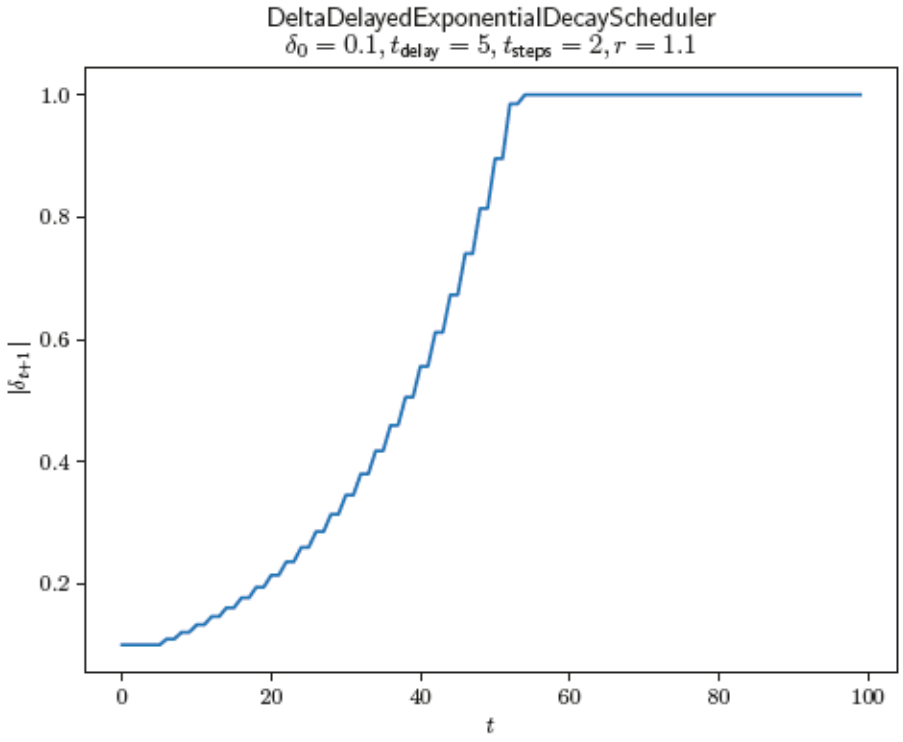


Figure 2.4: An example of a delayed exponential decay schedule of δ, using the calculation in Equation (2.7) with parameters $\delta_0 = 0.1$, $r = 1.1$, $t_{\text{delay}} = 5$ and $t_{\text{steps}} = 2$.

2.3.3 Post-training Process for Logical Interpretation

After training, the neural DNF model is automatically translated to an ASP program for logical interpretation. As proposed in pix2rule [28], this translation process is composed of three steps: pruning, thresholding, and translation, as shown in Figure 2.5.

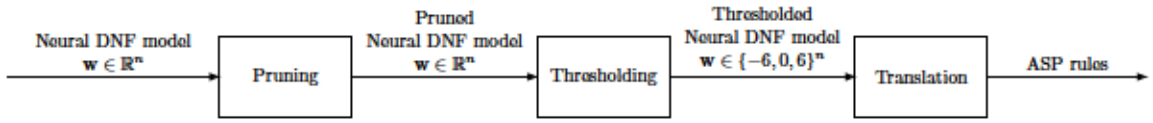


Figure 2.5: The post-training process to extract ASP rules from a neural DNF model.

Pruning. This step prunes the weights of the semi-symbolic node to make the model as sparse as possible without significantly changing model performance. Each edge is checked in turn to see if its removal changes the model output for any input, and it is removed if it is not significantly degrading the original model behaviour. The change is measured by the difference in the model's performance (for example, accuracy in pix2rule) on the training set, and it is considered significant if the difference is greater than a hyperparameter threshold τ_{prune}. Edges are checked in terms of their accumulated impact on the model output (i.e. the model parameters are updated at each check), and edges connecting conjunctive nodes to disjunctive nodes are checked first and then those connecting inputs to conjunctive nodes. At each time step t, the pruning process is formalised as follows:

$$w_i' = g_{\text{prune}}(w_i, \tau_{\text{prune}}) = \begin{cases} 0 & \text{if } J_{\text{metric}}(\mathbf{w}_t, \mathbf{X}_{\text{train}}) - J_{\text{metric}}(\mathbf{w}_{t,\text{with } w_i=0}, \mathbf{X}_{\text{train}}) < \tau_{\text{prune}} \\ w_i & \text{otherwise} \end{cases}$$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t,\text{with } w_i=w_i'}$$

where $J_{\text{metric}}(\cdot,\cdot)$ is a metric function that measures the model's performance, and the higher the value the better the performance. This process can be either executed once on the model, or the weights can be passed in order for multiple passes until no weights are removed. The complexity of the pruning process depends on both the size of the model and the number of inputs. The pruning process is formalised in Algorithm 1.

Thresholding. This process prepares the semi-symbolic nodes for logical translation, by converting the weights from $\mathbb{R}$ to values in $\{-6, 0, 6\}$. $\pm$ 6 is chosen in order to saturate tanh. All weights with absolute values smaller than the threshold τ_{thresh} are filtered out, while the rest are converted to ± 6 according to their sign. The thresholding process is formalised as the following:

$$g_{\text{threshold}}(\mathbf{w}; \tau_{\text{thresh}}) = \hat{\mathbf{w}}, \text{ with } \hat{w}_i = 6 \cdot \text{sign}(w_i) \cdot \mathbb{1}(|w_i| > \tau_{\text{thresh}})$$

where $\mathbb{1}(\cdot)$ is the indicator function and $0 \leq \tau_{\text{thresh}} \leq \max|\mathbf{w}|$. The best value τ^*_{thresh} is chosen by trying all possible values in the range (with step size 0.01) that maintains the model's behaviour as much as possible, subject to the same checks in pruning, and is applied across all conjunctive and

disjunctive nodes in the neural DNF model:

$$\forall \tau_{\text{thresh}} \in [0, \max|\mathbf{w}|, 0.01].$$
$$[J_{\text{metric}}\left(g_{\text{threshold}}(\mathbf{w}; \tau^*_{\text{thresh}}), \mathbf{X}_{\text{train}}\right) \geq J_{\text{metric}}\left(g_{\text{threshold}}(\mathbf{w}; \tau_{\text{thresh}}), \mathbf{X}_{\text{train}}\right)]$$

$J_{\text{metric}}(\cdot, \cdot)$ is the same metric function as before, and the notation $[a, b, s]$ denotes the set $\{a, a+s, a+2s, \ldots\}$ of values from a to b with step size s. The complexity of the thresholding process depends on the size of the set $|[0, \max|\mathbf{w}|, 0.01]| = \lfloor \max|\mathbf{w}|/0.01 \rfloor + 1$, and the dataset size since it needs to compute the metric function for each threshold value. The thresholding process is formalised in Algorithm 2.

Translation. This step translates the weights of the semi-symbolic node to logic programs. Conjunctive nodes are translated to a single rule. Disjunctive nodes are translated to multiple rules with the same rule head, with each rule having exactly one atom corresponding to its conjunctive node input. Any weight $w_i \in \{-6, 6\}$ indicates the presence of an atom in the rule body. This process is formalised and proved later in this thesis (Section 4.2.2).

Example 2.3.1: Post-training Process

Consider a neural DNF model with three inputs x_1, x_2, x_3, three conjunctive nodes c_1, c_2, c_3, and three disjunctive nodes d_1, d_2, d_3. Suppose after training the model has learned the following weight matrices, where $\mathbf{W}_{\text{c}} \in \mathbb{R}^{3\times3}$ denotes the weights from inputs to conjunctive nodes, and $\mathbf{W}_{\text{d}} \in \mathbb{R}^{3\times3}$ denotes the weights from conjunctive nodes to disjunctive nodes:

$$\mathbf{W}_{\text{c}} = \begin{pmatrix} 2.8 & -3.1 & 0.2 \\ -2.9 & 3.2 & 2.7 \\ 0.1 & 0.05 & -3.0 \end{pmatrix}, \quad \mathbf{W}_{\text{d}} = \begin{pmatrix} 3.5 & 0.1 & 0.0 \\ 0.0 & -2.8 & 3.1 \\ 3.3 & 0.0 & -2.9 \end{pmatrix}$$

Row i of $\mathbf{W}_{\text{c}}$ contains the weights connecting inputs x_1, x_2, x_3 to conjunctive node c_i, and row j of $\mathbf{W}_{\text{d}}$ contains the weights connecting conjunctive nodes c_1, c_2, c_3 to disjunctive node d_j.

Pruning. Suppose the pruning process with threshold $\tau_{\text{prune}} = 0.01$ determines that some weights can be removed without significantly affecting performance and they are set to zero, yielding:

$$\mathbf{W}_{\text{c}}^{\text{pruned}} = \begin{pmatrix} 2.8 & -3.1 & 0 \\ -2.9 & 3.2 & 2.7 \\ 0 & 0 & -3.0 \end{pmatrix}, \quad \mathbf{W}_{\text{d}}^{\text{pruned}} = \begin{pmatrix} 3.5 & 0 & 0 \\ 0 & -2.8 & 3.1 \\ 3.3 & 0 & 0 \end{pmatrix}$$

Thresholding. Suppose the thresholding process determines that the best threshold is $\tau^*_{\text{thresh}} = 2.7$, and it converts weights to strictly ±6 based on their sign and if their absolute values are greater than τ^*_{thresh}, yielding:

$$\mathbf{W}_{\text{c}}^{\text{thresh}} = \begin{pmatrix} 6 & -6 & 0 \\ -6 & 6 & 0 \\ 0 & 0 & -6 \end{pmatrix}, \quad \mathbf{W}_{\text{d}}^{\text{thresh}} = \begin{pmatrix} 6 & 0 & 0 \\ 0 & -6 & 6 \\ 6 & 0 & 0 \end{pmatrix}$$

Translation. Conjunctive nodes in $\mathbf{W}_{\mathrm{c}}^{\mathrm{thresh}}$ are translated to a single rule, while disjunctive nodes in $\mathbf{W}_{\mathrm{d}}^{\mathrm{thresh}}$ are translated to multiple rules with the same rule head, where each rule has exactly one atom corresponding to its conjunctive node input. A weight value of 6 indicates the presence of a positive atom in the rule body, while a weight value of -6 indicates the presence of a negative atom in the rule body. The detailed translation process is defined in Section 4.2.2.

$$\mathtt{d}_1 \leftarrow \mathtt{c}_1. \qquad \mathtt{c}_1 \leftarrow \mathtt{a}_1, \mathtt{not}\ \mathtt{a}_2.$$
$$\mathtt{d}_2 \leftarrow \mathtt{not}\ \mathtt{c}_2. \qquad \mathtt{c}_2 \leftarrow \mathtt{not}\ \mathtt{a}_1, \mathtt{a}_2.$$
$$\mathtt{d}_2 \leftarrow \mathtt{c}_3. \qquad \mathtt{c}_3 \leftarrow \mathtt{not}\ \mathtt{a}_3.$$
$$\mathtt{d}_3 \leftarrow \mathtt{c}_1.$$

Algorithm 1 Pruning of a Semi-symbolic Node

Input: Trained weights $\mathbf{w}$ of a semi-symbolic node, training set $\mathbf{X}_{\text{train}}$, metric function J_{metric}, and threshold τ_{prune}
Output: Pruned weights $\mathbf{w}$

procedure PRUNEEDGES($\mathbf{w}, \mathbf{X}_{\text{train}}, \tau_{\text{prune}}$)
  $J_{\text{base}} \leftarrow J_{\text{metric}}(\mathbf{w}, \mathbf{X}_{\text{train}})$
  for each edge weight w_i in $\mathbf{w}$ **do**
    $\mathbf{w}' \leftarrow \mathbf{w}$ with w_i set to 0
    $J_{\text{new}} \leftarrow J_{\text{metric}}(\mathbf{w}', \mathbf{X}_{\text{train}})$
    if $J_{\text{base}} - J_{\text{new}} < \tau_{\text{prune}}$ **then**
      $\mathbf{w} \leftarrow \mathbf{w}'$ ▷ Accumulate the removal
      $J_{\text{base}} \leftarrow J_{\text{new}}$
  return $\mathbf{w}$
▷ Edges into disjunctive nodes are pruned before edges into conjunctive nodes
$\mathbf{w}_{\mathrm{d}} \leftarrow$ PRUNEEDGES($\mathbf{w}_{\mathrm{d}}, \mathbf{X}_{\text{train}}, \tau_{\text{prune}}$)
$\mathbf{w}_{\mathrm{c}} \leftarrow$ PRUNEEDGES($\mathbf{w}_{\mathrm{c}}, \mathbf{X}_{\text{train}}, \tau_{\text{prune}}$)
▷ Optionally repeat until no further weights are removed

2.4 Reinforcement Learning

Reinforcement Learning (RL) [150] is a paradigm of learning where an agent learns to make decisions by interacting with an environment and with the objective of maximising the cumulative reward over time. Unlike supervised learning where a model learns from labelled examples, or unsupervised learning where a model discovers patterns in data without labels, RL agents learn through trial and error, receiving feedback in the form of rewards or penalties for their actions.

This section provides a brief overview of the fundamentals of RL (Section 2.4.1) and the actor-critic methods (Section 2.4.2) used in this thesis.

2.4.1 Fundamentals

RL tasks are commonly modelled as Markov Decision Processes (MDPs) [127] or sometimes Partially Observable Markov Decision Processes (POMDPs) [171, 80]. A finite MDP is defined by a tuple

Algorithm 2 Thresholding of Semi-symbolic Nodes

Input: Pruned weights $\mathbf{w}$ of all semi-symbolic nodes, training set $\mathbf{X}_{\text{train}}$, metric function J_{metric}, step size $s = 0.01$
Output: Thresholded weights $\hat{\mathbf{w}} \in \{-6, 0, 6\}$

procedure THRESHOLD($\mathbf{w}, \tau_{\text{thresh}}$)
  $\hat{w}_i \leftarrow 6 \cdot \text{sign}(w_i) \cdot \mathbb{1}(|w_i| > \tau_{\text{thresh}})$ for each $w_i \in \mathbf{w}$
  return $\hat{\mathbf{w}}$
procedure FINDBESTTHRESHOLD($\mathbf{w}, \mathbf{X}_{\text{train}}$)
  $J^* \leftarrow -\infty$, $\tau^*_{\text{thresh}} \leftarrow 0$
  for $\tau_{\text{thresh}} \in [0, \max|\mathbf{w}|, s]$ **do**
    $\hat{\mathbf{w}} \leftarrow$ THRESHOLD($\mathbf{w}, \tau_{\text{thresh}}$)
    $J \leftarrow J_{\text{metric}}(\hat{\mathbf{w}}, \mathbf{X}_{\text{train}})$
    if $J \geq J^*$ **then**
      $J^* \leftarrow J$, $\tau^*_{\text{thresh}} \leftarrow \tau_{\text{thresh}}$
  return τ^*_{thresh}
$\tau^*_{\text{thresh}} \leftarrow$ FINDBESTTHRESHOLD($\mathbf{w}, \mathbf{X}_{\text{train}}$)
$\hat{\mathbf{w}} \leftarrow$ THRESHOLD($\mathbf{w}, \tau^*_{\text{thresh}}$) ▷ Apply across all conjunctive and disjunctive nodes
return $\hat{\mathbf{w}}$

$\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, where $\mathcal{S}$ is the finite set of states, $\mathcal{A}$ is the finite set of actions, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability function ($\Delta(X)$ denotes the probability simplex over a finite set X, i.e. the set of all probability distributions over X), $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. The transition probability function p and reward function r model the environment dynamics, and are typically unknown, and the agent learns them (implicitly or explicitly) through interaction with the environment. At each time step t, an agent (i) observes the current state s_t of the environment, (ii) selects an action a_t based on its policy, and (iii) receives a reward r_t and transitions to a new state s_{t+1}, as illustrated in Figure 2.6. The Markov property states that the future state depends only on the current state and action, and not on the history of states and actions:

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots)$$
$$r(s_t, a_t) = r(s_t, a_t, s_{t-1}, a_{t-1}, \ldots)$$

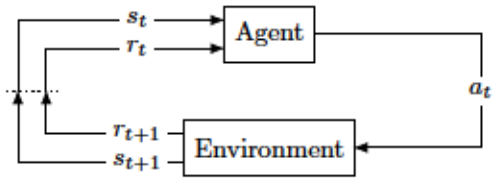


Figure 2.6: The agent-environment interaction in an MDP [150].

The goal of the agent is to learn a policy that maps states to action probabilities, i.e. $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, that maximises the expected cumulative reward over time, where the expected cumulative reward for some policy π is defined as:

$$J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t] \tag{2.8}$$

where $\gamma \in [0,1]$ is a discount factor that determines how much the agent values future rewards compared to immediate ones. The discount factor ensures convergence of the infinite sum by making distant rewards contribute less to the total, emphasising immediate rewards as more valuable than future ones. It also provides mathematical tractability for optimisation algorithms. When γ is close to 0, the agent focuses primarily on immediate rewards and acts short-sightedly. And when γ is close to 1, the agent considers long-term consequences more, potentially leading to more strategic behaviour.

In some cases, where the agent does not have access to the complete state information, the task is modelled as a Partially Observable MDP (POMDP), where the agent receives observations that provide partial information about the true state. Despite POMDPs' Markovian dynamics, since there is no direct access to the true state, the agent in general needs to keep track of past history to make optimal decisions.

2.4.2 Actor-Critic Methods

Two fundamental paradigms exist for learning policies in RL: value-based methods learn a value function that estimates expected returns and derive a policy from it, while policy-based methods directly parameterise and optimise the policy itself. Actor-critic algorithms such as Advantage Actor-Critic (A2C/A3C) [107] and Proximal Policy Optimization (PPO) [138] combine the benefits of both approaches by maintaining an actor that learns the policy $\pi(a_t|s_t) = \Pr(A_t = a_t|S_t = s_t, \pi)$ and a critic that learns the value function $V^\pi(s)$ for estimating the expected cumulative reward from each state. Before diving into the details of the actor-critic methods, it is helpful to understand the issues of purely value-based and purely policy-based methods.

Purely Value-Based Methods

Given a policy π, a value function $V^\pi(s)$ estimates the expected cumulative reward from a state s under the policy π, while a state-action value function $Q^\pi(s,a)$ estimates the expected cumulative reward from a state s and action a under the policy π.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[R_t|s_t = s] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \\ Q^\pi(s,a) &= \mathbb{E}_\pi[R_t|s_t = s, a_t = a] \\ &= \sum_{s'\in\mathcal{S}} p(s'|s,a)[r(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s')Q^\pi(s',a')] \end{aligned}$$

Value-based methods, such as Q-learning [159] and Deep Q-Networks (DQN) [108], learn the action-value function $Q^\pi(s,a)$ and derive an implicit policy from the value function (commonly an ϵ-greedy policy, where a random action is selected with probability ϵ and the action with the highest value is selected with probability $1-\epsilon$). The estimated Q-function $\hat{Q}(s,a)$ is updated using the Bellman equation:

$$\hat{Q}(s_t,a_t) \leftarrow \hat{Q}(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{a'\in\mathcal{A}} \hat{Q}(s_{t+1},a') - \hat{Q}(s_t,a_t)]$$

where α is the learning rate. For finite MDPs, tabular Q-learning is guaranteed to converge asymptotically to the optimal action-value function $Q^*(s,a)$ with probability 1 under standard assumptions [159]. DQN extends Q-learning by using deep neural networks to approximate the Q-function, enabling RL to handle high-dimensional state spaces. DQN also employs two techniques to improve learning stability and performance:

- **Experience replay**: storing and randomly sampling past transitions to break correlations in sequential data and improve data efficiency.
- **Target networks**: maintaining a separate *target network* with a copy of the Q-network's parameters to compute the target values, where the target network's parameters are only updated periodically with the Q-network's parameters. This decouples the target computation from the Q-network, reducing oscillations and improving training stability.

However, value-based methods struggle in continuous action spaces where computing the maximum action value is computationally intractable. Furthermore, because of its assumption of deterministic optimal policy given a known value function, it is not suitable to learn stochastic policies and might struggle in partially observable environments.

Purely Policy-Based Methods

Policy-based methods, such as REINFORCE [160], directly parameterise and optimise the policy $\pi(a_t|s_t)$. They have the advantages of natural handling of continuous action spaces, the ability to learn stochastic policies, and better convergence properties in certain scenarios. The update signal for REINFORCE relies on the estimated return (or the Monte Carlo return) G_t to update the policy parameters:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[G_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$
$$\text{where } G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Since the Monte Carlo return is a sum of rewards over a long horizon, it has high variance due to (1) the accumulation of variance from stochastic rewards along the trajectory and (2) attributing all future rewards to a single action despite those rewards depending on subsequent actions and states, leading to unstable learning [137].

Actor-critic, A2C and PPO

The actor-critic method [151] directly resolves the flaws of purely value-based and purely policy-based methods, by combining the parameterisation and optimisation of a policy (actor) with a value function (critic). The actor learns a policy directly, making it suitable for continuous action spaces and stochastic policies. Simultaneously, the critic learns a value function, which provides a low-variance feedback signal that replaces the high-variance Monte Carlo return, thus stabilising the actor's learning. In practice, actor-critic algorithms perform on-policy updates: they collect data by following the current policy, then update both the policy and the value estimator using the collected data. The feedback from the critic, typically the temporal-difference (TD) error $(r_t + \gamma V(s_{t+1}) -$

$V(s_t)$) or an advantage estimate, which quantifies how much better an action is compared to the average action in a given state, guides the gradient update direction of the actor. By using the value estimates as a baseline, the actor's updates become more stable than pure policy gradient. The actor and critic often share underlying network layers such as CNNs for state representation, and the architecture diverges into separate output layers for policy and value predictions after the shared layers.

A3C [107] and A2C [115] are two popular actor-critic methods that use the advantage function in their policy gradient update to reduce the variance of the update signal. The advantage function is defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

In practice, actors interact with multiple copies of the environment and collect trajectories. A2C is a synchronous variant of A3C, where all actors share the same policy and value function, in contrast to A3C's asynchronous updates where each actor has its own policy and value function. After a certain number of time steps or episodes, the experiences from all actors are aggregated. The algorithm then uses these to perform an update of actors and critics. The policy update uses the policy gradient with the advantage function $\hat{A}(s_t, a_t)$ as a weight:

$$\nabla_{\theta} J = \mathbb{E}_{\pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \hat{A}(s_t, a_t) \right]$$

The advantage estimates can be computed in various ways; a common choice is n-step returns or Generalised Advantage Estimation (GAE) [137] to balance bias and variance. Using GAE, the k-step advantage is computed as: $\hat{A}(s,a) = \sum_{t=0}^{k-1} \gamma^t r_{t+t} + \gamma^k V(s_{t+k}) - V(s_t)$.

PPO [138] is another actor-critic method with the key innovation of clipping the policy update to prevent destructively large changes. It builds upon Trust Region Policy Optimisation (TRPO) [136] but takes a simpler approach. TRPO constrains the policy updates by bounding the KL divergence between the old policy $\pi_{\theta_{\text{old}}}$ and the updated policy π_{θ}, either as an explicit constraint or as a penalty term, to ensure stability. Rather than including the KL-divergence term, PPO employs a clipped objective ($L^{\text{CLIP}}(\theta)$ in Equation (2.10)) that discourages the new policy from deviating too far from the old one. By replacing the KL constraint with clipping, PPO can be optimised using standard first-order gradient methods, avoiding the second-order computations (such as Hessian-vector products) required by TRPO. This makes PPO simpler to implement while achieving comparable or better performance. Beyond algorithmic simplicity, code-level optimisations have also been shown to contribute to PPO's improved performance over TRPO [47].

A typical PPO training iteration alternates between sampling and optimisation. First, the current actor is used to collect trajectories from the environment (often using parallel copies of the environment for efficiency, as in A2C). The returns and advantage estimates are then computed from these trajectories. Finally, PPO constructs the surrogate objective and optimises the parameters over mini-batches for several epochs. The PPO objective function contains three terms: the difference between the predicted and target value functions ($L^{\text{value}}(\theta)$), the difference between the new and old policies ($L^{\text{CLIP}}(\theta)$), and a bonus term for the entropy of the policy ($S[\pi_{\theta}](s_t)$, to encourage exploration). The PPO loss function used in this thesis is based on the original PPO paper

[138] and CleanRL's implementation [72], defined as:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) + \kappa_1 L^{\text{value}}(\theta) - \kappa_2 S[\pi_\theta](s_t) \right] \tag{2.9}$$

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{2.10}$$

$$S[\pi_\theta](s_t) = -\mathbb{E}_t \left[\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \right],$$

$$L^{\text{unclipped v}}(\theta) = (V_\theta(s_t) - V_t)^2,$$

$$V^{\text{clipped}}(\theta) = V_{\text{old}}(s_t) + \text{clip}(V_\theta(s_t) - V_{\text{old}}(s_t), -\epsilon, \epsilon),$$

$$L^{\text{clipped v}}(\theta) = (V^{\text{clipped}}(\theta) - V_t)^2,$$

$$L^{\text{value}}(\theta) = \mathbb{E}_t \left[\max \left(L^{\text{unclipped v}}(\theta), L^{\text{clipped v}}(\theta) \right) \right]$$

where:

- $r_t(\theta)$ is the probability ratio between the new and old policies at time t;
- κ_1 and κ_2 are hyperparameters and the coefficients of the value loss and the entropy;
- π_θ is the policy of the model with parameter θ;
- a_t is the action at time t;
- s_t is the state at time t;
- $\hat{A}_t$ is the advantage estimate at time t, computed with GAE [137];
- $\text{clip}(x, a, b)$ is a function that clamps x to the range $[a, b]$;
- ϵ is a hyperparameter, representing the clipping range;
- $S[\pi_\theta](s_t)$ is the entropy of the policy in state s_t;
- $V_\theta(s_t)$ is the value predicted in state s_t by the model with parameter θ, during the loss calculation;
- V_t is the value target at time t;
- and $V_{\text{old}}(s_t)$ is the value predicted in state s_t by the model with parameter θ, during the collection of experiences.

In practice, the expectations $\mathbb{E}_t[\cdot]$ are approximated by taking the sample mean over timesteps from collected trajectories. Notably, the original PPO paper [138] proposes an alternative to the clipped probability ratio objective as $L^{\text{CLIP}}(\theta)$ in Equation (2.10). The variant uses a KL-divergence penalty in the objective and adjusts a penalty coefficient to keep the KL divergence between the new and old policy below a target. In practice, the clipped version has become more popular due to its simplicity and empirical success.

With the clipping mechanism, PPO achieves better performance because of the stability it provides. Furthermore, PPO has empirically better sample complexity than traditional policy gradient methods that perform single-pass updates [138]. A2C performs only one update for each batch of on-policy trajectories. In contrast, PPO updates the policy multiple times per batch of data, giving it an advantage in how effectively it uses each sample.

Chapter 3

Related Work

This chapter surveys research relevant to neural DNF-based models, covering in particular the learning of interpretable rules and their applications in reinforcement learning. Section 3.1 examines neuro-symbolic learning approaches that combine neural networks with symbolic reasoning for rule learning. Section 3.2 discusses end-to-end neuro-symbolic reinforcement learning for interpretable policy learning and other RL tasks using hybrid architectures.

3.1 Neuro-symbolic Learning

This section focuses on neuro-symbolic learning approaches. It begins with purely differentiable methods that enable end-to-end training whilst maintaining symbolic interpretability in Section 3.1.1. Section 3.1.2 examines other hybrid architectures that combine separate symbolic and neural components where the symbolic components are the core rule learners. Section 3.1.3 reviews predicate invention from raw data in the context of both purely differentiable and hybrid architectures. Finally, Section 3.1.4 discusses approaches that enforce logical constraints on neural network outputs, which show similarities with the design of the neural DNF-EO model proposed in Chapter 5.

3.1.1 End-to-end Rule Learning

Many existing rule learning works take the Inductive Logic Programming (ILP) [110, 29] from symbolic AI as their core frameworks and realise the ILP tasks with neural networks for end-to-end differentiable rule learning. Classical ILP systems, such as ALEPH [147], Metagol [30], ILASP [92] and FastLAS [91], aim to find the best set of rules in a hypothesis space that, when combined with a given background knowledge, would cover the target examples, i.e. entail all positive examples but none of the negative ones. The learned rules are human-readable because of their symbolic structure. ILP systems are able to generalise well even with a small amount of data, as most systems prefer to choose the shortest set of rules in the hypothesis space that cover the given examples. It is also possible for some systems not to choose the shortest rules: FastLAS [91] supports a custom scoring function that evaluates hypotheses and learns the lowest-scoring hypothesis, making it possible for the system to choose less general but more relevant domain-specific solutions. Despite these advancements, ILP systems suffer from scalability when the task has a huge hypothesis space, requiring a large amount of memory and time to generate and search for optimal solutions. The

scalability issue hinders the ILP systems' capability in real-world datasets where most tasks have large hypothesis spaces. FastLAS [91] is designed to be more scalable by computing a subset of the hypothesis space while guaranteeing to cover the optimal solution, yet it can still suffer from time-out issues. On the other hand, neural networks are more capable of handling large datasets, but at the cost of interpretability. Differentiable ILP approaches combine the best of ILP and neural networks to create systems that are interpretable and scalable. In general, differentiable ILP approaches approximate discrete operations and input with continuous operations and input, and create a process of continuous logic reasoning such that the model can be trained via gradient descent. Some differentiable ILP approaches focus on solving symbolic or knowledge base tasks that benefit much from having background knowledge in the systems and thus inheriting the merit of ILP systems, while other methods inject little to no prior knowledge into the models, thus avoiding the human-engineering overhead. These existing methods can be categorised into two classes: template-based methods that restrict the forms of the rules to learn and free-form methods that do not require predefined templates.

Template-based Methods

Many differentiable ILP approaches follow the use of rule templates in classical ILP to restrict the search space. A *rule template* (or a *meta-rule*) is a second-order clause, for example:

$$\mathrm{H}(X,Y) \leftarrow \mathrm{B}_1(X,Z) \wedge \mathrm{B}_2(Z,Y) \tag{3.1}$$

where H, B_1 and B_2 are predicate variables. A rule constructed from a rule template has to follow the number of body atoms and each predicate's arity specified in the template, but it has the freedom of choosing which predicate to use. For instance, $Q(X,Y) \leftarrow R(X,Z) \wedge S(Z,Y)$ is an instance of the rule template in Equation (3.1), where Q, R and S are specific binary predicates. Methods that use rule templates essentially inject a bias into the model of what form of rules to learn, which helps with scaling and interpretability of the model.

δ-ILP [50] serves as the foundation for many differentiable ILP systems. It generates a set of clauses from rule templates and reduces the inductive learning task to a satisfiability problem by assigning a Boolean flag to each clause to indicate whether it should be included in the final rule set. To enable differentiable training, the system replaces discrete logic inputs and operations with continuous counterparts. During inference, rather than using Gödel's or Łukasiewicz's t-norm, which suppresses the gradient of one conjunct, the product t-norm is used, serving as the fuzzy conjunction operator, ensuring that forward chaining remains differentiable. Given the initial valuation of a set of ground atoms, δ-ILP performs multiple forward-chaining steps on the generated clauses and predicts the probability of a label. The objective is to match the predicted label with the ground truth by minimising the negative log-likelihood loss. Each intensional predicate (i.e. predicates that appear in the head of the rules) is assumed to be defined by exactly two clauses. Consequently, a weight matrix is assigned to each pair of clauses, with dimensions equal to the product of the number of clauses generated by the pair of rule templates that define the intensional predicate. Different rule templates yield weight matrices of different dimensions. Each element in the matrix represents the confidence that the corresponding pair of clauses correctly defines the intensional predicate. These weights constitute the only trainable components in the system. After training, softmax is applied to

the weights to obtain a probability distribution over clauses, and clauses above a confidence threshold are selected as solutions to the ILP task. Compared to Metagol [30], a symbolic ILP system, δ-ILP achieves similar performance on tasks with binary predicates and can learn in circumstances where Metagol times out, demonstrating the superior scalability of differentiable systems. The limitation is its expressivity: the learnable clauses are constrained by the rule templates, which are restricted to contain no constants and exactly two body atoms with predicates of arity at most two.

LRI [23] introduces a neural forward-chaining theorem prover that learns logical rules by representing rule predicates and facts as dense embeddings trained via soft unification. The model encodes each rule as a tuple of predicate embeddings and variable slots, and each fact as a predicate embedding with subject-object constants and a real-valued valuation in $[0,1]$. Forward chaining applies all rules to all fact pairs for a fixed number of steps, generating new candidate facts whose valuations are computed as products of fact values and cosine similarities between rule and fact predicates. For ILP tasks, predicate embeddings for rules are optimised by binary cross-entropy on target valuations, while for theory learning tasks the valuations of all candidate facts are also learned and regularised with an L2-like sparsity penalty on initial facts to encourage compression of the observation set. Different from δ-ILP, LRI operates at the level of atom embeddings rather than explicit symbolic rule search: both rules and invented predicates are interpretable by inspecting their learned embeddings, enabling predicate invention and fact induction within a single differentiable framework. However, the approach still relies on hand-specified rule templates and suffers from exponential growth of inferred facts in forward chaining.

HRI [64] proposes a proto-rule, which extends the notion of meta-rule to enable more expressive representation, while using soft unification for predicate matching. The predicate variable in the body of a proto-rule can be instantiated with any predicate with arity equal to or lower than the variable's arity, and the arguments of the predicate variable can also be optional if the actual predicate used has lower arity than the variable. The head predicate's arity must match the variable's arity. For example, the proto-rule $\mathrm{H}(X,Y) \leftarrow \overline{\mathrm{B}_1}(X,Z) \wedge \overline{\mathrm{B}_2}(Z,Y)$ can be instantiated with the predicate $Q(X,Y) \leftarrow R(X) \wedge S(Z,Y)$, where Q and S are binary predicates and R is a unary predicate. Using proto-rules, HRI proposed a hierarchical model for differentiable ILP, by organising auxiliary predicates generated by proto-rules in a sequence of layers. The auxiliary predicate can be defined by predicates from the layers before, and the current layer if recursion is allowed. The last layer's predicates are matched with the target predicate. Before training, the model is instantiated with a set of proto-rules, and at each iteration, binary cross entropy encourages the final layer's valuation to be close to the ground truth valuation. Like LRI [23], the learned embeddings of predicate and atom in the meta-rule are matched via soft unification measured with cosine similarity. The model performs multiple steps of forward chaining to infer the valuation of the predicates and computes a loss between the inferred values and the ground truth. To avoid the underestimation problem due to the multiplication of values in $[0,1]$, HRI replaces the multiplication for a conjunction with a min operator and softmaxes the cosine unification score with Gumbel noise across the possible ground atoms for the same rule atom. After training, the highest unification score predicates are selected to form the rules at each layer. In comparison with δ-ILP [50] and LRI [23], HRI [64] achieves similar performance with a smaller and more flexible template set: the same proto-rule set used by HRI is generic enough for all the experiments carried out in [64], which covers the entire set of function-free Horn clauses with at most 2 body atoms that involve unary and binary predicates.

LNN-ILP [140] introduces a differentiable ILP framework that learns both first-order logic rules and their logical connectives using Logical Neural Network (LNN) with tree-style templates. The architecture contains individual components of sparsely associating parameters with predicates and LNN-conjunction/disjunction for fact combination. The LNN parameters are trained end-to-end under linear inequality constraints. However, the framework still relies on user-specified templates and full grounding of the knowledge base, which may limit automation and scalability on very large or perception-heavy tasks.

Neural Theorem Prover (NTP) [132] uses Prolog's backward chaining as guidance to recursively construct neural networks, performing multi-hop reasoning over knowledge bases while learning sub-symbolic embeddings. The method replaces discrete unification with a differentiable similarity function over predicate and constant embeddings, and decomposes proving into differentiable OR and AND modules that operate on proof states containing both a substitution set and a continuous proof success score. NTP uses rule templates to instantiate rule structures and the rules' parameters are updated via gradient descent based on the success signal of proving known or unknown facts. NTP is limited to relatively small knowledge bases due to the combinatorial explosion of differentiable proof paths and relies on predefined rule templates and function-free logic, constraining the expressivity of induced programs. [106] addresses the computational complexity by approximating the proof search with nearest-neighbour retrieval, enabling application to larger knowledge bases. The new approach, NTP 2.0, prunes the computation graph aggressively at each unification step but keeps the same architecture and optimisation setup as NTP. It reduces the problem of unifying a query with all facts in the knowledge base to a k-nearest neighbours problem in the embedding space, and replaces exhaustive continuous unification in the OR module with approximate nearest neighbour search. Greedy Neural Theorem Prover (GNTP) [105] goes further with computation optimisation and also extends to natural language. GNTPs cast fact selection for each sub-goal as an approximate nearest-neighbour search in embedding space and similarly prune rule expansions by grouping rules into structural templates and only unifying with nearby rule heads in predicate-embedding space. It introduces an attention mechanism over known predicate embeddings to parameterise rule templates more compactly, and a simple but effective bag-of-embeddings text encoder so that logical predicates and textual surface patterns share a common embedding space, enabling joint reasoning over triples and sentences. Its reliance on approximate nearest-neighbour heuristics may miss useful proof paths, and the bag-of-embeddings text encoder ignores word order, potentially losing important information.

In summary, the approaches described above show great performance with interpretability and scalability using rule templates to help restrict the search space and reduce computation. However, the choice of rule templates requires careful human engineering to ensure a solution would be included in the hypothesis space, and thus the templates set is usually task-specific. Although HRI [64] alleviates the task-specific problem to some extent by using proto-rules, which are more flexible, its generic proto-rule set still limits the form of rules a model can learn, and still needs to be modified if applied to a more complex environment with higher-arity predicates or more body atoms.

Another way to constrain the search space is to use mode declarations [109]. αILP [143] proposes a differentiable ILP system that learns rules from images using mode declarations. The system combines a pre-trained object-centric visual perception module with a facts converter that maps detected objects and attributes into probabilistic facts via neural predicates, and a differentiable

forward-chaining engine that performs multi-step inference using weighted clauses. Clause structure is learned by a top-k beam search over clause refinements guided by a differentiable evaluation score, while clause weights are optimised with gradient descent using a soft program composition that treats logic programs as smooth combinations of clause functions. The learned models yield human-readable first-order classification rules, where the weight distributions sharply peak on a single dominant clause. However, αILP is memory-intensive due to the large clause-fact index matrices and assumes a pre-trained perception model, plus carefully designed mode declarations that encode background knowledge, with performance dependent on search depth and beam size.

NEUMANN [144] improves αILP by using bipartite graphs for better memory efficiency. The method compiles weighted first-order definite clauses into a single bipartite forward reasoning graph whose nodes represent ground atoms and conjunctions, thus being more memory efficient than constructing the full clause-grounding matrices. It then performs differentiable forward chaining via bidirectional message passing that softly implements logical conjunction and disjunction over probabilistic atom valuations. Clauses are learned through an iterative loop that alternates between gradient-based optimisation of clause weights via cross-entropy and differentiable sampling using the Gumbel-max trick and mode declarations. Apart from memory efficiency, NEUMANN also demonstrates that its sampling loop generates and refines clauses faster than beam search over candidate clauses without sacrificing accuracy. However, it is still bounded by the usage of mode declarations.

The approaches discussed above demonstrate alternative strategies using mode declarations for constraining the hypothesis space. However, these methods remain dependent on carefully designed mode declarations, limiting their generalisability across different tasks the same way rule templates do.

Free-form Methods

Many approaches do not rely on rule templates to restrict the hypothesis space. Instead, they use neural operations to approximate logical ones and train generic neural networks without prior bias on the form of rules. The weights of the networks can be used as indications of atoms' presence in the rules, if the method provides any interpretability.

Neural Logic Machine (NLM) [43] learns logic rules in a multi-layer multi-group neural architecture. Grounded function-free predicates are represented as probabilistic matrices with values in the range $[0, 1]$, and the model uses neural units to approximate boolean logic and logic quantifiers. The multi-layer multi-group architecture realises the forward chaining by inter-group and intra-group computations. Inter-group computation takes the previous layer's output and its neighbouring groups' outputs, and then aligns their dimensions with expansion and reduction operators. Intra-group computation realises boolean logic with MLPs and sigmoid activations, taking the aligned outputs from inter-group computation as input. The number of forward chaining steps is limited by the number of layers the NLM has. In comparison to δ-ILP, NLM displays the same performance but with better scalability: the computational complexity of NLM scales polynomially with respect to the number of predicates, whereas δ-ILP scales exponentially. Furthermore, NLM does not require rule templates to restrict the form of rules learned. However, NLM is not fully interpretable because of the use of MLPs.

dNL-ILP [123] is another approach that converts the inductive problem to a SAT problem by

finding the best membership function, similarly to δ-ILP. In contrast to δ-ILP, dNL-ILP assigns a trainable boolean flag m_i on each individual atom, indicating whether the atom is in the rule or not. It also introduces conjunctive and disjunctive neurons that implement neural conjunction and disjunction functions to enable differentiable training. The model consists of a conjunctive layer followed by a disjunctive layer, representing a neural disjunctive normal form. At each inference step, a neural conjunctive/disjunctive function is applied to all possible grounded body atoms of a predicate, and this value is amalgamated with the input value of this predicate to be the new input at the next timestamp. Such a process is repeated as the forward chaining process, and the output is interpreted as a probability. dNL-ILP suffers from scaling issues as the complexity grows exponentially with the number of atoms in the body. The multiplication of values in $[0, 1]$ in its fuzzy logic operations may also lead to the vanishing gradient problem.

Differentiable Logic Machine (DLM) [170] builds upon NLM's architecture and improves on its interpretability issue. Logic modules that implement fuzzy logic replace the MLP, and half of the outputs are defined with fuzzy conjunction while the other half use fuzzy disjunction. Weights are used to choose predicates to satisfy the constraints, and conjunction and disjunction do not share the same weights. The fuzzy logic operations can be replaced by boolean counterparts to extract logic programs after training. In addition, truth and falsity constants are added to allow a layer to be preserved for the later layer. DLM shows good performance and better scalability without sacrificing interpretability, and also does not require rule templates. However, the overall architecture design is complicated, with the manual placement of negation applied to half of the conjunction's and disjunction's outputs.

The work in [142] introduces a differentiable ILP framework that overcomes the structural and expressivity limits of δ-ILP by enabling learning from noisy, structured data with function symbols and multi-clause programs. The approach constructs its search space with an example-guided beam search starting from a hand-crafted generic clause. Its backward-chaining fact-enumeration algorithm restricts ground atoms to those necessary for inference, thus controlling the size of the clauses. The method softly composes candidate programs using multiple clause-level weights and a smooth OR operator, avoiding the quadratic parameter explosion of pairwise clause weighting. Experiments show that the approach reduces memory usage and runtime and results in fewer clauses, but also show the reliance on effective search-space pruning and difficulty scaling to highly expressive domains.

Several works build upon the Learning from Interpretation Transition (LFIT) framework [74], which learns logic programs from state-transition data, to propose different differentiable ILP frameworks. D-LFIT [57] learns interpretable propositional transition rules from noisy or incomplete state-transition data in a differentiable manner. It transfers symbolic logic programs into differentiable vector spaces with an immediate vector consequence operator. It then uses a two-stage architecture: a meta-info learner that learns a sparse normal matrix encoding the rules of the logic program to reduce the search space, and an interpretation learner that, given the meta-information, optimises same head variable (SHV) matrices under product t-norm semantics to recover the connections between variables and logic operators. Once trained, D-LFIT extracts logic programs simply by thresholding and decoding SHV matrices, yielding linear-time rule extraction. DFORL [56] extends D-LFIT [57] to learn first-order clauses. The model still focuses on learning same-head programs and reuses the differentiable immediate-consequence computation of D-LFIT, with the addition of

auxiliary-predicate invention and similarity losses that encourage rule diversity and curriculum learning by treating prior knowledge as constraints. It also introduces a sampling pre-processing strategy to select essential facts for better scalability when learning from knowledge graphs. Both works suffer from scalability issues as the number of body variables increases, causing combinatorial growth in matrix dimensions. Besides learning transition rules, δLFIT2 [125] also focuses on realising the symbolic property of variable permutation invariance in a differentiable manner. The model takes the head of the rule, the body length of the rule and the state transition as input, and outputs the probabilities of a set of rules with that head and body length. A novel rotating permutation function reorders the variables to generate variable assignments and their embeddings are then fed into a set transformer followed by linear layers to produce the probabilities of the rules, and there is a dedicated output layer for each body length. The approach scales to larger systems by training on fixed-size subsets of variables and sliding this window over the Herbrand base, though this still requires separate models per maximum rule body length and suffers from combinatorial growth in rule outputs.

Neural Logic Network (NLN) [124] addresses the challenge of learning accurate yet interpretable rule-based models that can recover the underlying logic programs governing tabular data. The neurons act as probabilistic AND and OR operators, and the biases are used to handle unobserved data. Each rule is constructed as if-then modules with shared binary, categorical, and continuous input encodings, enabling parameter-efficient learning while preserving logical structure. NLN is trained via gradient descent with tricks such as regularisation, rule resets, weight discretisation, pruning, and bias re-adjustment to yield sparse and human-readable rules. Due to the formulation of the neurons, NLN can suffer from vanishing gradients, and the weight discretisation becomes computationally expensive when the number of rules increases. The many training tricks also introduce many levels of randomness, making it difficult to fully characterise the performance of the model.

Pix2rule [28] addresses the problem of free-form rule learning through a simple formalisation of a semi-symbolic layer and formulates a neural DNF model. The background of the neural DNF model is described in Section 2.3. In particular, all experiments in pix2rule [28] are binary classifications that can be rendered indirectly as ILP tasks with no background knowledge: the model tries to learn a set of rules that predict whether the target is true or not.

Table 3.1 summarises the two categories of differentiable ILP approaches mentioned in this section and their key limitations.

This thesis leverages the semi-symbolic layer proposed in pix2rule [28] but goes beyond synthetic binary classification tasks, proposing a family of neural DNF-based models suitable for multi-class/multi-label classification and reinforcement learning tasks. Compared to methods that rely on rule templates (e.g. δ-ILP [50], HRI [64], LNN-ILP [140], NTP variants [132, 106, 105]) or mode biases (e.g. αILP [143], NEUMANN [144]), the neural DNF-based models developed in this thesis and the neural DNF model from pix2rule [28] are not restricted to learning rules in certain forms, and can approximate a larger hypothesis search space. The factors affecting the form of learned rules are the input and the number of outputs from the conjunctive layer: the input directly affects the types of atoms or grounded predicates and the maximum number of them that can be included in a conjunct, while the number of conjunctive layer outputs affects the number of disjuncts in a DNF formula. While having a larger hypothesis space, it is still able to learn in a short time through gradient descent, showing better scalability than symbolic learners like FastLAS [91]. The logical

Method	Bias	Key Limitation
δ-ILP [50]	Rule templates	Task-specific templates and restricted to $\leq$2 body atoms of arity $\leq$2
LRI [23]	Rule templates	Exponential growth of inferred facts during forward chaining
HRI [64]	Proto-rules	Expressivity still bounded by proto-rule arity
LNN-ILP [140]	Tree templates	Requires full grounding of the knowledge base
NTP [132]	Rule templates	Combinatorial explosion of differentiable proof paths
NTP 2.0 [106]	Rule templates	Approximate nearest-neighbour search may miss useful proof paths
GNTP [105]	Rule templates	Bag-of-embeddings encoder ignores word order and heuristic pruning may miss proofs
αILP [143]	Mode declarations	Memory-intensive and assumes a pre-trained perception model
NEUMANN [144]	Mode declarations	Expressivity bounded by mode declarations
NLM [43]	None	MLPs prevent full interpretability of learned rules
dNL-ILP [123]	None	Exponential complexity in number of body atoms and susceptible to vanishing gradients
DLM [170]	None	Complex architecture with manual placement of negation
D-LFIT [57]	None	Combinatorial growth of matrix dimensions with body variables and propositional only
DFORL [56]	None	Combinatorial matrix growth and scalability degrades with many body variables
δLFIT2 [125]	None	Requires separate models per rule body length
NLN [124]	None	Susceptible to vanishing gradients and weight discretisation is computationally expensive
Pix2rule [28]	None	Restricted to binary classification settings

Table 3.1: Comparison of differentiable ILP approaches surveyed in this section. Bias column represents the type of search-space constraint.

bias in the semi-symbolic layers allows the neural DNF-based model to be interpreted after post-training process via logical translation. The formulation of the semi-symbolic layer is simpler than the neural conjunctive/disjunctive functions in dNL-ILP [123] and NLN [124] and is able to alleviate the vanishing gradient problem. Overall, the proposed neural DNF-based models with semi-symbolic layers show a high degree of interpretability and scalability as differentiable rule learning models with simplicity in architecture design.

3.1.2 Rule Learning with Hybrid Architectures

Apart from the category of differentiable inductive logic programming, which focuses on developing a neural model that enables back-propagation and symbolic rule extraction, there are other neuro-symbolic learning approaches with separate symbolic and neural components. These methods do not allow full back-propagation due to their symbolic components, but compute gradients from the symbolic components' outputs in various ways to train the neural component.

One foundational work is DeepProbLog [101], which extends ProbLog, a probabilistic logic programming language, to interface with deep learning. It introduces Neural Annotated Disjunctions, which connect neural network outputs via softmax probabilities to logical atoms in a probabilistic program. By computing gradients through all proof paths of the ground truth label using ProbLog's gradient semi-ring, DeepProbLog enables differentiable training, where error signals propagate from the symbolic reasoning component with handcrafted rules back to the neural network. Its improvement, DeepProbLogA* [102], uses A* search to find the most likely proof path according to a heuristic function, thus improving the speed. It also employs an exploration mechanism to avoid getting stuck in local minima.

Another work, $Meta_{Abd}$ [33], learns Datalog rules with an abduction-induction procedure, relying on the Metagol learning system [30] as its symbolic component. The model is trained with

Expectation Maximisation algorithm, where the maximisation step is a gradient descent algorithm on the neural network. It shows better performance on MNIST arithmetic tasks than a fully neural architecture while providing interpretable logic rules, as well as better scalability than DeepProbLog. However, the choice of Datalog limits expressiveness to definite logic (no functions, constraints, etc.).

Answer Set Programming [95] is another popular logic programming language with strong expressiveness. NeurASP [165] interprets network outputs as probabilities over symbolic atoms, and formalises a semantics that assigns such probabilities to ASP stable models. It derives gradients using semantic loss [164] for end-to-end training through symbolic constraints. The original NeurASP work focuses on neuro-symbolic reasoning; that is, the rules for reasoning are not learned but encoded in advance, and only the perception network is trained. Nevertheless, it lays the foundation for other works on rule learning (see NSIL [32] below). However, it has a potential scalability issue from stable-model enumeration for large or highly combinatorial programs.

FFNSL [31] and NSIL [32] use ILASP [92] or FastLAS [91] as their noise-tolerant symbolic learner. The FFNSL method converts neural predictions into symbolic features using a Data-to-Knowledge generator, attaching confidence-weighted facts to each example for the LAS learner. It relies on a fixed and pre-trained network whose prediction failures can still dominate and increase the symbolic solve time under high noise. NSIL improves on the neural learning task by jointly learning a neural perception model and ASP programs from raw data. The system formulates perception as latent concept classification and uses NeurASP-style semantic loss [165] for training neural outputs consistent with the symbolic constraints from the symbolic learner. The weights of the examples for the LAS learner are iteratively updated through corrective examples that balance exploration and exploitation during symbolic search. The scalability issue is inherited from NeurASP and might be further exacerbated by the bootstrap stage when generating the initial hypothesis.

NeuralFastLAS [26] presents another neuro-symbolic learning framework that extends FastLAS [91] to enable simultaneous learning of symbolic knowledge and neural networks. The network training uses semantic loss [164] to train the two heads: the perception head that learns latent concepts, and the rule probability head that learns the posterior distribution on the rules. Nevertheless, it inherits several constraints from the underlying FastLAS system, including limited support for non-observational predicate learning.

Embed2Rule [13] learns first-order logic programs directly from raw perceptual data by decoupling neural perception learning from symbolic rule induction. The neural pipeline first trains an end-to-end neural perception-and-reasoning model to produce clustered latent embeddings, and then assigns optimal symbolic labels to these clusters with the Hungarian Algorithm, using weak labels from a zero-shot vision-language model (VLM). The symbolic component ILASP [92] induces answer set programs from the noisy symbolic labels. The use of VLM for weak cluster labelling is novel but it can also be problematic when the VLM struggles with specific visual domains.

Being an ASP-based approach, the work in [49] shows another interesting way of training two separate components in a pipeline rather than constructing loss functions for gradient back-propagation on the neural network. It uses a Binary Neural Network (BNN) to translate raw sensory data to a sequence of disjunctions and an apperception engine [51] to learn symbolic theory to explain the sequence with strong inductive bias. The theory searching algorithm of the apperception engine is encoded as ASP itself and solved with an ASP solver, and the training of BNN is encoded as a logic program as well. So the entire training process is encoded into a single ASP task.

Method	Symbolic Component	Key Limitation
DeepProbLog [101]	ProbLog	Scalability limited by proof-path enumeration
DeepProbLogA* [102]	ProbLog	Heuristic search may miss optimal proofs
$Meta_{Abd}$ [33]	Metagol	Expressivity limited to definite Datalog (no functions or constraints)
NeurASP [165]	ASP solver	Scalability limited by stable-model enumeration
FFNSL [31]	ILASP / FastLAS	Fixed perception network and prediction errors exacerbate symbolic solve time
NSIL [32]	FastLAS	Inherits NeurASP scalability issues and bootstrap stage may inflate hypothesis space
NeuralFastLAS [26]	FastLAS	Limited support for non-observational predicate learning
Embed2Rule [13]	ILASP	Weak cluster labelling degrades in specialist visual domains
BNN-AE [49]	ASP solver	Separate pipeline with no gradient flow

Table 3.2: Comparison of hybrid neuro-symbolic rule learning approaches surveyed in this section.

3.1.3 Neural-Symbolic Predicate Invention

One challenge of neuro-symbolic systems is the need for symbolic predicates. They can either be already available in the task or hand-crafted by experts when the input data is not structured, for example, images or text. However, hand-crafted predicates can be costly and time-consuming to create, and may not generalise well to other tasks. Therefore, it is desirable to automatically invent predicates from raw data. Approaches with hybrid architectures in Section 3.1.2 address this with explicit neural components that may or may not be trained, while this section reviews some additional works that tackle this challenge.

Deep Symbolic Learning (DSL) [34] jointly learns perception functions and symbolic functions from raw images. It models each perception module as a neural network producing symbol confidence distributions and injects discrete symbolic choices through a differentiable policy mechanism that selects symbols while still enabling gradient-based learning. The symbolic function, stored as a learnable matrix of weights, maps all possible combinations of symbols to the target concept. DSL computes the truth value of predictions using Gödel t-norm semantics and optimises the minimum confidence across the selected perception and symbolic choices, ensuring that only the least confident internal decision is updated.

NeSy-π [141] automatically invents relational concepts from visual scenes. It iteratively generates clause candidates from basic neural predicates, evaluates them using differentiable reasoning, and scores each clause set via necessity and sufficiency metrics computed in a 2D scoring space derived from positive-negative scene pairs. New predicates are composed by taking disjunctions of high-scoring clause bodies, and these invented predicates are fed back into a NeSy-ILP solver that optimises weighted clauses via gradient descent. Experiments show that NeSy-π outperforms αILP without background using limited primitive predicates.

Predicate invention is equally important in neuro-symbolic RL approaches, where the agent's observations are often unstructured inputs rather than symbolic representations. Section 3.2.1 discusses some RL methods that face this challenge, such as NeSyRL [84], which employs a pre-trained semantic parser to extract first-order predicates from text observations. The neural DNF-MT model proposed in this thesis enables end-to-end predicate invention by allowing preceding neural layers to learn higher-level representations from unstructured observations during training.

3.1.4 Network Learning with Logical Constraints

Different from learning rules as differentiable ILP methods, some approaches focus on enforcing constraints on the neural network's outputs. While not directly related, the work in Chapter 5 shows design similarities such as encoding logical constraints as layers.

DL2 [53] introduces a unified way to inject and query logical knowledge into neural networks by translating first-order style constraints into non-negative, almost-everywhere differentiable losses. These losses are exactly zero when the constraints are satisfied, enabling both constraint-guided training and declarative queries over inputs, outputs, and neurons. The method has additional training overhead and is susceptible to local minima.

Semantic loss [164] is used for the gradient back-propagation from symbolic constraints to the neural network. It penalises neural outputs that do not match the semantics of the symbolic constraints encoded as circuits. However, the loss does not guarantee consistency and often acts as a regulariser rather than a constraint. An extension to the semantic loss in [4] adds an entropy term to improve its consistency but still cannot guarantee it.

SPL [3] introduces Semantic Probabilistic Layers (SPLs), a drop-in predictive layer that makes neural networks' structured outputs both probabilistic and guaranteed to satisfy rich logical constraints. SPL combines an expressive conditional probabilistic circuit with a compiled constraint circuit to enforce hard consistency. However, computing the product of two circuits and renormalising could be wasteful, and its simplification, single-circuit SPLs, may be over-parameterised and may slow training down.

C-HMCNN(h) [62] addresses hierarchical multi-label classification by enforcing the logical constraint of implication between classes from the task to improve predictive performance. The method wraps a base network h with a max-constraint module, which takes the network's output and for each class computes the maximum score over its subclasses, guaranteeing no hierarchy violations by construction. A max-constraint loss modified from standard BCE encourages delegating superclass decisions to informative subclasses and avoids bad optima, which are common with standard BCE. Its extension CCN$^+$ [63] improves on the constraint that can be encoded and converts arbitrary propositional-logic requirements into rules via resolution. A dependency graph is built from the rules to enable assignment of multiple labels to the same levels, which are then translated to requirement layers on top of a base network. The requirement layers clamp each label within a coherence interval derived and the whole model is trained with a tailored requirements loss so gradients flow to the right labels without inducing wrong supervision. CCN$^+$ shows better performance than neural baselines without violations and its effectiveness is demonstrated in a safety-critical autonomous-driving benchmark. However, the approach can be sensitive to label ordering and also susceptible to size blow-ups from saturation or unfavourable interaction graphs.

3.2 Neuro-symbolic Reinforcement Learning

This section reviews existing neuro-symbolic reinforcement learning approaches. Section 3.2.1 focuses on neuro-symbolic RL approaches that perform interpretable policy learning, which is closely related to the neural DNF-MT model introduced in Chapter 6. Section 3.2.2 summarises some wider neuro-symbolic RL approaches that do not necessarily perform policy learning but other RL tasks such as reward learning and constraint learning.

3.2.1 Policy Learning for RL

Several end-to-end neuro-symbolic approaches have been proposed for interpretable policy learning in reinforcement learning settings, formalising the task as a differentiable ILP task. They are commonly applied to Relational RL [44, 167] domains, which utilise symbolic representations for states, actions, and policies. Depending on the differentiable ILP system used for policy learning, some approaches require rule templates or mode declarations to restrict the search space while others do not.

NLRL [77] combines δ-ILP [50] with the policy gradient method to learn policies in first-order logic, improving interpretability and cross-task generalisation. NLRL proposes a Differentiable Recurrent Logic Machine (DRLM) that extends δ-ILP with clause-level softmax weights, resulting in lower memory use. DRLM performs iterative differentiable deduction and assigns softmax-normalised weights to candidate clauses generated from rule templates and composes one-step deduction functions to propagate truth values. The actor uses DRLM at its core and wraps it with a symbolic state encoder that converts state to atoms first and a differentiable action decoder that maps atom valuations to action probabilities after DRLM's deduction. The overall policy is trained with vanilla policy gradient and a tailored action decoder normalisation encourages interpretable, high-confidence action rules rather than brittle relative-score hacks. NLRL shows promising results on block-manipulation and cliff-walking variants, with near-optimal performance and better robustness to unseen scenarios than MLP baselines. However, the reliance on hand-crafted rule templates and no raw-perception pipeline for state encoding brings human engineering overhead, and it shows occasional sub-optimality in induced rules.

NUDGE [39] bridges the gap between interpretability and explainability in RL using the method proposed in [142]. By distilling a pre-trained neural policy into a differentiable first-order logic policy where weighted rules describe and justify actions, the policy becomes both readable and state-contingently explainable while remaining competitive in performance. It converts object-centric states into probabilistic logical atoms, performs multi-step differentiable forward reasoning over candidate action rules, and extracts an action distribution with a smooth log-sum-exp aggregator. Rule candidates are generated from mode declarations efficiently via agreements with the neural oracle, and rule weights are optimised with a PPO-style actor-critic update. NUDGE shows comparable or better performance, faster convergence, and better robustness to environment variants than purely neural PPO. However, its dependency on sufficiently expressive predicates and good coverage of the neural guide constrains the completeness or biases the rule search.

NLM [43] is tested on Blocks World, where the agent takes actions on an initial world such that it would eventually match a target world. The agent only receives a positive reward when it completes the task. The NLM outputs relational predicates, and an extra fully connected layer is added to take the concatenated relational representation of object pairs. The model is trained to predict the effect of the actions rather than an action probability, with a loss consisting of REINFORCE loss and a cross-entropy loss to help with exploration. Inherited from the NLM's architecture, the model itself remains not fully interpretable.

NeSyRL [84] employs a Logical Neural Network (LNN) for policy learning without rule templates or mode declarations. An LNN [131] constrains the weights of neurons to mimic logical AND and OR in the form of a recurrent neural network. Each neuron of an LNN represents an element of a logical formula, either an atom (concept) or a connective (operator). The LNN outputs probability bounds for atoms being true after multiple passes, and in addition to the training objective loss, a

contradiction loss is used to penalise contradictions in the system. A pre-trained semantic parser extracts first-order predicates from observation text, using atom classes to select the appropriate LNN. The LNN then selects actions to generate trajectories that get stored in a replay buffer for training, similar to DQN [108]. At each time step, actions are computed using class-selected LNNs with converted first-order logic, and the final action is determined by arg max-ing over all propositional atoms. The method shows significantly faster convergence and the use of LNN makes it possible to extract symbolic policy, but with the limitation of relying on a pre-trained semantic parser for predicate invention.

DLM [170] associates weights with predicates rather than rules, thus not relying on rule templates or mode declarations. DLM builds upon NLM [43] to realise forward chaining, but with logical computation units to provide interpretability. A DLM actor is trained with actor-critic PPO [138], with a specially designed critic with GRUs to handle different-arity predicates, such that it can take predicates as input and estimate the value of the current state. The critic uses r GRU heads for arity r predicates, where each head processes objects at specific positions, and outputs cascade from higher to lower arities. The final output combines arity 1 GRU results with nullary predicates, feeding into an MLP for state value prediction. The actor consists of two separate parts of reasoning and decision-making. The reasoning component provides a vector of actions, and the decision-making component predicts a probability distribution over the actions; the final action is selected through softmax. Compared to NLM, DLM is faster and more memory efficient, and can extract interpretable policies for simple tasks in Blocks World setting due to its logical units. However, no interpretable policy can be extracted from DLM under complex tasks due to many local minima when enforcing interpretability, and the specially crafted critic is not generalisable to other tasks.

Method	Bias	Obs. Type	RL Algorithm
NLRL [77]	Rule templates	Symbolic	Policy gradient
NUDGE [39]	Mode declarations	Symbolic extracted by pre-trained networks raw observations	PPO
NLM [43]	None	Symbolic	REINFORCE
NeSyRL [84]	None	Symbolic extracted by pre-trained networks text	DQN-style
DLM [170]	None	Symbolic	PPO
Neural DNF-MT (this thesis)	None	Symbolic from environment; or extracted by trainable preceding layers from raw observations	PPO

Table 3.3: Comparison of end-to-end neuro-symbolic approaches for interpretable policy learning. Bias column represents the type of search-space constraint used. Obs. Type column represents the type of observations consumed by the agent.

Different from NLRL [77] and NUDGE [39], the neural DNF-MT model proposed in this thesis does not use rule templates or mode declarations. Therefore, it does not rely on human engineering to construct the inductive bias and can learn a wider range of rules. Compared to the works that either operate on relational-based observations [77, 39, 170] or require pre-trained networks to extract logical predicates [39, 84], the neural DNF-MT model is shown to be end-to-end trainable with preceding layers for predicate invention from scratch. Compared to DLM [170], the neural

DNF-MT model is trained via PPO but with a simple MLP critic, and the learned policy can always be extracted.

Complementary to the declarative, logic-based approaches discussed above, another family of neuro-symbolic RL methods represents policies as explicit programs rather than first-order rules. PROPEL [154] and REVEL [9] learn programmatic control sequences using a *lift-update-project* pipeline: the policy is lifted into a continuous neural representation for gradient-based updates and then projected back onto a discrete program space via imitation learning, subject to structural constraints from the program grammar. GALOIS [24] takes a different approach, introducing a sketch-based domain-specific language that combines declarative and imperative programming paradigms and learning white-box policies via differentiable ILP. These methods yield interpretable programmatic policies and support formal verification and cross-task generalisation. However, they share a key limitation with the logic-based approaches above: they rely on hand-crafted or pre-processed state representations, such as relational knowledge bases, rather than learning predicates end-to-end from raw observations.

3.2.2 Hybrid Models with Classic ILP for RL

Many existing works utilise classic ILP systems for learning not only policies but also rewards, constraints, or heuristics.

An example is the body of work based on the induction of reward machines. A reward machine [73] is a formalisation of reward functions in RL, taking the form of a finite-state machine that assigns a reward to each state. It is an effective way to model non-Markovian rewards in RL by encoding subgoals into edges. HRM [55] learns hierarchies of reward machines that are able to call each other to improve modularity and reusability. Each call is treated as a subtask and learned via a curriculum learning approach. PROB-IRM [121] presents a learning method to learn RMs from noisy traces with a reward shaping technique using posterior Bayesian beliefs, better suited for real-world scenarios when the labelling function is not perfect. FORM [12] proposes the learning of RMs with first-order logic formulae that label RM transitions, which improves the compactness and transferability of RMs. All these works approach the RM learning as ILP tasks and use ILASP [92] as a learner, and the policy learning is typically done with (deep) Q-learning or some related methods.

Neuro-symbolic Inverse Constrained Reinforcement Learning (NSICRL) [38] integrates a deep Q-learner and an ILP system (ALEPH [147]) to learn policies and defeasible constraints simultaneously. The system shows the ability to infer weighted, high-level logical constraints that are not only transferable across different environments but also support lifelong learning by allowing new information to override previously learned constraints. However, its effectiveness depends on the quality of expert demonstrations and a pre-existing knowledge base provided as the background knowledge to the ILP system.

The work in [156] tackles sample inefficiency and lack of interpretability in Q-learning by learning human-readable heuristics online with ILP and using them to directly guide exploration rather than reward shaping. Specifically, each training batch's top-return episodes are converted into weighted examples for FastLAS [91] to learn human-readable policy heuristics. During exploration, the agent samples suggested actions from the ASP solver with a probability derived from the average return of saved episodes and otherwise acts uniformly, seeding the very first batch with a greedy heuristic

from the first episode to avoid cold starts.

Method	ILP System	NeSy Task	RL Algorithm
HRM [55]	ILASP	Reward machine learning (hierarchical)	Q-learning
PROB-IRM [121]	ILASP	Reward machine learning (noisy)	Q-learning
FORM [12]	ILASP	Reward machine learning (FOL)	Q-learning
NSICRL [38]	ALEPH	Defeasible constraint learning	Q-learning
Veronese et al. [156]	FastLAS	Online heuristic learning	Q-learning

Table 3.4: Comparison of hybrid neuro-symbolic RL approaches that use classic ILP systems for structured knowledge induction. NeSy Task column represents the RL task addressed symbolically.

These hybrid approaches demonstrate the versatility of combining classic ILP systems with neural RL components for learning structured knowledge such as rewards, constraints, and heuristics, with the common characteristic of using established ILP systems to induce symbolic knowledge from interactions and using neural components to handle the underlying policy or value-function learning. In comparison, the neural DNF-based models proposed in this thesis adopt a fully differentiable approach that learns policies.

Chapter 4

Semi-Symbolic Nodes and Neural DNF Model

The core methodology introduced in this thesis centres on the *semi-symbolic node* and the *neural disjunctive normal form (DNF) model*. Inspired by the foundational formulation in pix2rule [28], this thesis develops a family of neural DNF-based models with architectures designed similarly to an MLP, but whose learned knowledge can be translated into logical rules represented in ASP. Instead of linear nodes, it uses semi-symbolic nodes as basic computation units. These nodes employ an elegant mathematical formulation to achieve neuro-symbolic conjunctive or disjunctive behaviours over their inputs and mitigate the vanishing gradient problem introduced by using t-norm fuzzy logic [68]. By stacking a layer of conjunctive nodes followed by a layer of disjunctive nodes, the architecture mimics a logical DNF formula. Section 2.3 provides the relevant definitions and an overview of the semi-symbolic node and the neural DNF model. Pix2rule demonstrates the first application of this approach for learning rules in synthetic binary classification tasks; however, it does not further explore the properties of the semi-symbolic node or provide any theoretical guarantees.

This chapter first formalises the semi-symbolic node's behaviour and interpretation in Section 4.1. Section 4.2 then mathematically proves the crucial property of translation from a semi-symbolic node to a logic program. It also proposes and proves the novel ability to translate a logic program back to a semi-symbolic node, thus fully closing the bidirectional cycle.

This chapter includes material from our paper [16] and [17].

4.1 Formalisation of Semi-Symbolic Node

This section first introduces terminology for describing the behaviour and interpretation of semi-symbolic nodes in Section 4.1.1. These concepts are essential for defining the desirable properties that guide the design of the neural DNF-based models developed in this thesis. Section 4.1.2 then formalises the *discretisation* process, which is a key stage in the automatic post-training pipeline outlined in Section 2.3.3. Discretisation transforms a trained neural DNF model into a form suitable for logical translation by converting weights from a continuous range to the discrete set $\{-6, 0, 6\}^N$. This transformation is necessary because real-valued weights merely reflect each input's relative contribution to the node's output; without discretisation, they cannot be mapped to a discrete

bivalent logical form.

4.1.1 Interpretation and Soft-Valued Truth Table

Recall from Section 2.3.1 that a semi-symbolic node with trainable weights w_i, $i = 1, \ldots, N$ and parameter δ, under input $\mathbf{x} \in [-1, 1]^N$ is given by:

$$f_{\mathbf{w}}(\mathbf{x}) = \left(\sum_{i=1}^{N} w_i x_i + \beta \right) \tag{4.1}$$

$$\beta = \delta \left(\max_{i=1}^{N} |w_i| - \sum_{i=1}^{N} |w_i| \right) \tag{4.2}$$

$$\hat{y} = \tanh\left(f_{\mathbf{w}}(\mathbf{x})\right) \tag{4.3}$$

The parameter δ controls the node's logical behaviour: the node mimics a conjunction when $\delta = 1$ and a disjunction when $\delta = -1$. Pix2rule [28] restricts the input and output range of the node to $[-1, 1]$ or $(-1, 1)$ for the activation, with the intention of extreme values to be interpreted as $\bot$ and $\top$. **Generalised Belief Logic (GBL)** is introduced to formally interpret the semi-symbolic node's activation.

Definition 4.1.1 (Generalised Belief Logic)**.** *A neural DNF-based model that builds upon semi-symbolic layers represents a logic system. This logic system is referred to as* ***Generalised Belief Logic (GBL)****. A semi-symbolic node's activation $y_i \in (-1, 1)$ represents its belief in a logical proposition. For each activation y_i, a bivalent interpretation function $f_b : (-1, 1) \to \{\bot, \top\}$ maps the activation to its bivalent logical interpretation $b_i \in \{\bot, \top\}$:*

$$b_i = f_b(y_i) = \begin{cases} \top & \text{if } y_i > 0 \\ \bot & \text{otherwise} \end{cases}$$

Since the behaviour of a semi-symbolic node resembles a logical conjunction or disjunction, it can be described similarly to a logical formula. Just as a logical truth table describes the behaviour of a logical formula, the **soft-valued truth table** is defined for describing the behaviour of a semi-symbolic node, as illustrated in Example 4.1.1.

Definition 4.1.2 (Soft-valued Truth Table of Semi-symbolic Node)**.** *A soft-valued truth table captures the output behaviour of a semi-symbolic node with weights $\mathbf{w} \in \mathbb{R}^N$ over all possible bivalent inputs $\mathcal{X} = \{-1, 1\}^N$. Each row starts with a possible combination of input values $\mathbf{x} \in \mathcal{X}$, followed by the corresponding node's activation $\tanh(f_{\mathbf{w}}(\mathbf{x}))$ and bivalent interpretation $b = \top$ if $\tanh(f_{\mathbf{w}}(\mathbf{x})) > 0$ and $\bot$ otherwise.*

With both the bivalent logical interpretation and the soft-valued truth table, the behaviour of a semi-symbolic node can now be described more formally.

Example 4.1.1: Soft-valued truth table

Consider a trained conjunctive semi-symbolic node with weights $\mathbf{w} = [-6, -2, -2, 2, -6]$. Its bias is calculated as $\beta = \max|\mathbf{w}| - \sum|\mathbf{w}| = -12$. The soft-valued truth table of this node is shown below, with some rows where $b = \bot$ omitted for brevity.

x_1	x_2	x_3	x_4	x_5	Node activation $\tanh(\sum_{i=1}^{5} w_i x_i + \beta)$	Bivalent Interpretation b
1	1	1	1	1	-1.000	$\bot$
					...	
-1	1	-1	1	-1	0.964	$\top$
-1	-1	1	1	-1	0.964	$\top$
-1	-1	-1	1	-1	1.000	$\top$
-1	-1	-1	-1	-1	0.964	$\top$

4.1.2 Discretisation

To guarantee a bidirectional neural-bivalent-logic translation, which is introduced and proved later in Section 4.2, the weights of the semi-symbolic nodes must take certain discrete values. A process such as the thresholding process mentioned in Section 2.3.3 that performs this weight transformation is called the **discretisation process**.

Definition 4.1.3 (Discretisation Process for Semi-symbolic Node). *The discretisation process converts the semi-symbolic nodes' weights from a continuous range $\mathbb{R}^N$ to a discrete set $\{-6, 0, 6\}^N$.*

The discretisation process can introduce drastic changes to the neural model when converting the continuous weights to discrete values, especially when the weights have intermediate magnitudes (neither close to 0 nor ± 6), and thus part of the model's desired behaviour might be changed or lost if the discretisation process is not well-designed. This problem of model behaviour change caused by discretisation leading to performance degradation is observed across all experiments in this thesis. In particular, the intermediate-valued weights represent inputs that contribute to only some logical rules, not all — a phenomenon termed *entanglement*. Chapter 7 will discuss this entanglement problem and how it heavily affects the thresholding process in detail, and introduces a novel discretisation method that better preserves the model's behaviour. Apart from redesigning the discretisation process, auxiliary loss functions such as $|w||6-|w||$ (Figure 4.1) can be included during training to encourage weights towards $\{-6, 0, 6\}^N$, though this does not guarantee convergence to these values.

4.1.3 Properties of Discretised Semi-Symbolic Node

Once the discretisation process is applied, the weights of the semi-symbolic node are all in a discrete set $\{-6, 0, 6\}^N$. Some properties can be derived from these discretised semi-symbolic nodes.

Firstly, some definitions are introduced for the semi-symbolic node based on its weights.

Definition 4.1.4 (Relevant Set of Semi-symbolic Node). *The relevant set $\mathcal{J}_{\mathbf{w}}$ of a semi-symbolic node with weights $\mathbf{w} \in \mathbb{R}^N$ is defined as the set of indices of the weights that are non-zero.*

$$\mathcal{J}_{\mathbf{w}} = \{i \mid i \in \{1, \ldots, N\}, w_i \neq 0\} \tag{4.4}$$

Note that for a discretised semi-symbolic node, all elements in the relevant set have an absolute

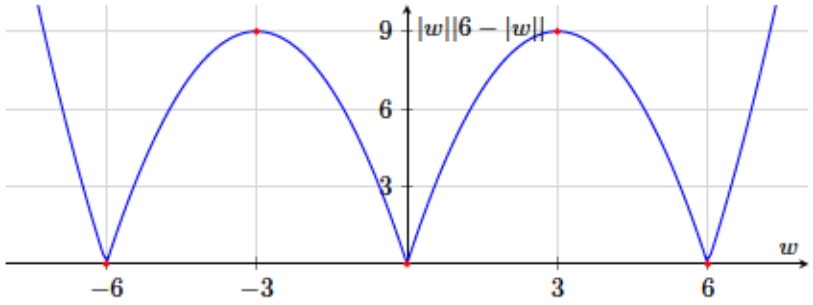


Figure 4.1: The auxiliary loss function $|w||6-|w||$ used to encourage weights to converge to the discrete set $\{-6, 0, 6\}$. The function reaches zero at the desired discrete values and has a maximum penalty at $w = \pm 3$. Compared to L_1 regularisation, $|w||6-|w||$ also allows ± 6 to be used, enforcing more than just sparsity.

value of 6. The notation $\mathcal{J}_{\mathbf{w}}$ can be simplified to $\mathcal{J}$ for brevity when the weights are clear from the context.

The relevant set can be further partitioned based on the signs of its elements. To formalise this, a general notational convention for sign-based index sets is introduced.

Definition 4.1.5 (Sign-based Index Set). *Given an index set $\mathcal{J}$ and a vector $\mathbf{a} \in \mathbb{R}^N$, the sign-based index set $\mathcal{J}_{\mathbf{a}}^{op}$ denotes the subset of indices in $\mathcal{J}$ whose corresponding elements in $\mathbf{a}$ satisfy the sign condition specified by the superscript $op \in \{+, -\}$.*

$$\mathcal{J}_{\mathbf{a}}^{+} = \{i \mid i \in \mathcal{J}, sign(a_i) = 1\} \tag{4.5}$$

$$\mathcal{J}_{\mathbf{a}}^{-} = \{i \mid i \in \mathcal{J}, sign(a_i) = -1\} \tag{4.6}$$

Definition 4.1.6 (Positive-sign and Negative-sign Relevant Sets). *Applying the sign-based index set convention to the relevant set $\mathcal{J}_{\mathbf{w}}$ on the weights $\mathbf{w}$ of a semi-symbolic node, the positive-sign relevant set $\mathcal{J}_{\mathbf{w}}^{+}$ and the negative-sign relevant set $\mathcal{J}_{\mathbf{w}}^{-}$ are defined as follows.*

$$\mathcal{J}_{\mathbf{w}}^{+} = \{i \mid i \in \mathcal{J}_{\mathbf{w}}, sign(w_i) = 1\} \tag{4.7}$$

$$\mathcal{J}_{\mathbf{w}}^{-} = \{i \mid i \in \mathcal{J}_{\mathbf{w}}, sign(w_i) = -1\} \tag{4.8}$$

By construction, $\mathcal{J}_{\mathbf{w}}^{+} \cap \mathcal{J}_{\mathbf{w}}^{-} = \emptyset$ and $\mathcal{J}_{\mathbf{w}}^{+} \cup \mathcal{J}_{\mathbf{w}}^{-} = \mathcal{J}_{\mathbf{w}}$.

When analysing the behaviour of a semi-symbolic node with respect to a specific input, it is useful to consider how the signs of the weights relate to the signs of the corresponding input elements. This motivates the following definitions of sign-matching and sign-mismatching sets, which extend the sign-based index set notation to capture the relationship between two vectors.

Definition 4.1.7 (Sign-matching Set between Semi-symbolic Node and Input). *Given a semi-symbolic node with weights $\mathbf{w} \in \mathbb{R}^N$ and an input $\mathbf{x} \in \{-1, 1\}^N$, the sign-matching set $\mathcal{J}_{\mathbf{w} \odot \mathbf{x}}^{+}$ contains all indices in the relevant set where the weight and input share the same sign. Equivalently, it is the positive-sign index set of the element-wise product $\mathbf{w} \odot \mathbf{x}$, restricted to the relevant set $\mathcal{J}_{\mathbf{w}}$.*

$$\mathcal{J}_{\mathbf{w} \odot \mathbf{x}}^{+} = \{i \mid i \in \mathcal{J}_{\mathbf{w}}, sign(w_i \cdot x_i) = 1\} \tag{4.9}$$

Definition 4.1.8 (Sign-mismatching Set between Semi-symbolic Node and Input). *Given a semi-symbolic node with weights $\mathbf{w} \in \mathbb{R}^N$ and an input $\mathbf{x} \in \{-1,1\}^N$, the sign-mismatching set $\mathcal{J}^-_{\mathbf{w}\odot\mathbf{x}}$ contains all indices in the relevant set where the weight and input have opposite signs. Equivalently, it is the negative-sign index set of the element-wise product $\mathbf{w}\odot\mathbf{x}$, restricted to the relevant set $\mathcal{J}_{\mathbf{w}}$.*

$$\mathcal{J}^-_{\mathbf{w}\odot\mathbf{x}} = \{i \mid i \in \mathcal{J}_{\mathbf{w}}, sign(w_i \cdot x_i) = -1\} \tag{4.10}$$

By construction, for any $\mathbf{x} \in \{-1,1\}^N$, the sign-matching and mismatching sets are mutually exclusive and their union is equal to the relevant set, that is, $\forall \mathbf{x} \in \{-1,1\}^N . \mathcal{J}^+_{\mathbf{w}\odot\mathbf{x}} \cup \mathcal{J}^-_{\mathbf{w}\odot\mathbf{x}} = \mathcal{J}_{\mathbf{w}}$.

A key property of the *raw output* of a discretised semi-symbolic node under input $\mathbf{x} \in \{-1,1\}^N$ is that its value is always a multiple of 6 and cannot be 0. For a conjunctive node, the maximum of its pre-activation output is exactly 6 (Proposition 4.1.1), and for a disjunctive node, the minimum of its pre-activation output is exactly -6 (Proposition 4.1.2).

Proposition 4.1.1. *Consider a **discretised conjunctive** semi-symbolic node c with N inputs and $\delta = 1$, with weights $\mathbf{w}_c \in \{-6,0,6\}^N$ and at least one non-zero weight element (i.e. $|\mathcal{J}_{\mathbf{w}_c}| > 0$). Given an input $\mathbf{x} \in \{-1,1\}^N$, the node's raw output will never be 0, specifically, $f_{\mathbf{w}_c}(\mathbf{x}) = 6$ or $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6$.*

Proof. Because the node is discretised, all elements in the relevant set have an absolute value of 6, that is, $\forall i \in \mathcal{J}_{\mathbf{w}_c} . |w_{ci}| = 6$. From Equation (4.2), the bias and the raw output of this discretised conjunctive node can be calculated as:

$$\begin{aligned}
\beta &= \max_{i=1,\ldots,N} |w_{ci}| - \sum_{i=1}^{N} |w_{ci}| = 6 - \sum_{i \in \mathcal{J}_{\mathbf{w}_c}} 6 = 6 - 6|\mathcal{J}_{\mathbf{w}_c}| \\
f_{\mathbf{w}_c}(\mathbf{x}) &= \sum_{i=1}^{N} w_{ci} x_i + \beta \\
&= \sum_{i \in \mathcal{J}_{\mathbf{w}_c}} w_{ci} x_i + 6 - 6|\mathcal{J}_{\mathbf{w}_c}|
\end{aligned}$$

Since $\forall i \in \mathcal{J}_{\mathbf{w}_c} . [\text{sign}(w_{ci}) \neq 0, |w_{ci}| = 6, |x_i| = 1]$, this gives:

$$\forall i \in \mathcal{J}_{\mathbf{w}_c} . w_{ci} x_i = \begin{cases} 6 & \text{if sign}(w_{ci}) = x_i \\ -6 & \text{otherwise} \end{cases}$$

Thus, the raw output can be reformulated using the sign-matching and sign-mismatching sets:

$$\begin{aligned}
f_{\mathbf{w}_c}(\mathbf{x}) &= \sum_{i \in \mathcal{J}_{\mathbf{w}_c}} w_{ci} x_i + 6 - 6|\mathcal{J}_{\mathbf{w}_c}| \\
&= \sum_{i \in \mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}} 6 + \sum_{i \in \mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}} (-6) + 6 - 6|\mathcal{J}_{\mathbf{w}_c}| \\
&= 6 \left(\sum_{i \in \mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}} 1 + 1 - |\mathcal{J}_{\mathbf{w}_c}| \right) \\
&= 6 \left(|\mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}| - |\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}| + 1 - |\mathcal{J}_{\mathbf{w}_c}| \right) \\
&= 6 \left(1 - 2|\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}| \right) \qquad \text{given } |\mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}| + |\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}| = |\mathcal{J}_{\mathbf{w}_c}|
\end{aligned}$$

Since $|\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}| \in \mathbb{N}$, the upper bound of $1 - 2|\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}|$ is 1, and $(1 - 2|\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}|) \in \{1, -1, -3, -5, \ldots\}$. Thus, $f_{\mathbf{w}_c}(\mathbf{x}) \in \{6, -6, -18, -30, \ldots\}$.

■

Proposition 4.1.2. *Consider a* ***discretised disjunctive*** *semi-symbolic node d with N inputs and $\delta = -1$, with weights $\mathbf{w}_d \in \{-6, 0, 6\}^N$ and at least one non-zero weight element (i.e. $|\mathcal{J}_{\mathbf{w}_d}| > 0$). Given an input $\mathbf{x} \in \{-1, 1\}^N$ (which can be the activation output from a conjunctive layer), the node's raw output will never be 0, specifically, $f_{\mathbf{w}_d}(\mathbf{x}) = -6$ or $f_{\mathbf{w}_d}(\mathbf{x}) \geq 6$.*

The disjunctive case follows the same sign-matching analysis as Proposition 4.1.1, but uses the disjunctive bias from Equation (4.2) ($\delta = -1$); the resulting bound on the raw output is the mirror image of the conjunctive case. A full proof is given in Appendix A.1.

This property of the raw output value of a discretised semi-symbolic node means that the tanh activation will be saturated and be in $\{-1, 1\}$. Having the tanh activation saturated allows the discretised semi-symbolic nodes to be chained together: the activation output of a discretised node will always be discrete values $\{-1, 1\}$ and the same property will hold for the next discretised node taking the previous node's activation output as input.

4.2 Bidirectional Neural-Bivalent-Logic Translation

Once a semi-symbolic node is discretised, it can be translated to a bivalent logic program, represented as an ASP program. This section demonstrates that such a neural-bivalent-logic translation is bidirectional and maintains truth value equivalence between the node's soft-valued truth table and the ASP program's truth table. More precisely, the bivalent logical interpretation of the node given an input is the same as the truth value of the rule head evaluated with the ASP program of the rule and the facts, and vice versa. By establishing this equivalence in both directions, this work closes the bidirectional cycle between the neural DNF model and logical programs, thereby opening a new research direction of incorporating logical programs into neural architectures. Specifically, Chapter 6 illustrates the practical implications of this bidirectionality by showing how the neural DNF-MT model can be modified without retraining by directly altering its logical program.

To develop this bidirectional translation, this section first introduces the symbolisation function, then demonstrates how to translate a discretised semi-symbolic node to an ASP program and proves

the truth value equivalence. Subsequently, it presents the reverse translation from a restricted form of ASP programs to discretised semi-symbolic nodes, again with provable equivalence.

4.2.1 Symbolisation and Projection

To establish a correspondence between the numerical representations of a neural DNF model and the symbolic representations of a logic program, two fundamental functions are required. The first is a *symbolisation function*, which assigns symbolic names to the neural variables, thereby creating a vocabulary for the logic program. The second is a *projection function*, which translates the numerical values or activations of these variables into logical facts. Together, these functions form the bridge between the neural and symbolic domains, enabling the bidirectional translation that preserves truth value equivalence.

Definition 4.2.1 (Symbolisation Function)**.** *Consider:*

- *the set of variables of a neural DNF model, including input variables* $X_i \in \{-1,1\}, i \in \{1,\ldots,N_{input}\}$*, tanh-activated output variables* $Y_{c_i} \in (-1,1), i \in \{1,\ldots,N_{conjunctions}\}$*, where* c_i *is the* i*-th conjunctive node in the conjunctive layer, and tanh-activated output variables* $Y_{d_i} \in (-1,1), i \in \{1,\ldots,N_{disjunctions}\}$*, where* d_i *is the* i*-th disjunctive node in the disjunctive layer;*

- *a Herbrand universe* $HU = \{a_i | i \in \{1,\ldots,N_{input}\}\} \cup \{c_i | i \in \{1,\ldots,N_{conjunctions}\}\} \cup \{d_i | i \in \{1,\ldots,N_{disjunctions}\}\}$.

The symbolisation function σ *of a neural DNF model is a function that maps the neural variables to the constants in the Herbrand universe:*

$$\sigma(X_i) = a_i \qquad \text{for input variables} \tag{4.11}$$

$$\sigma(Y_{c_i}) = c_i \qquad \text{for output variables of conjunctive node } c_i \tag{4.12}$$

$$\sigma(Y_{d_i}) = d_i \qquad \text{for output variables of disjunctive node } d_i \tag{4.13}$$

The inverse of σ is denoted as σ^{-1}, and maps symbolic predicate names back to the original names of inputs or bivalent interpretations of the nodes.

Definition 4.2.2 (Projection Function)**.** *The projection function* φ_σ *of a neural DNF model is a function that maps the values or the activations of the neural variables to facts.*

$$\varphi_\sigma(X_i) = \sigma(X_i) \qquad \text{if } x_i = 1 \tag{4.14}$$

$$\varphi_\sigma(Y_{c_i}) = \sigma(Y_{c_i}) \qquad \text{if } y_{c_i} > 0 \tag{4.15}$$

$$\varphi_\sigma(Y_{d_i}) = \sigma(Y_{d_i}) \qquad \text{if } y_{d_i} > 0 \tag{4.16}$$

This function can be used to translate a vector to a set of facts. For input $\mathbf{x} \in \{-1,1\}^{N_{input}}$*, tanh-activated output of conjunctive node* c_i $\mathbf{y}_{c_i} \in (-1,1)^{N_{conjunctions}}$*, and tanh-activated output of disjunctive node* d_i $\mathbf{y}_{d_i} \in (-1,1)^{N_{disjunctions}}$*, the projection function* φ_σ *applies to each element of the*

vector and generates a set of facts:

$$\varphi_\sigma(\mathbf{x}) = \{\sigma(X_i) \mid i \in \{1,\dots,N_{input}\}, x_i = 1\} \tag{4.17}$$

$$\varphi_\sigma(\mathbf{y}_{c_i}) = \{\sigma(Y_{c_i}) \mid i \in \{1,\dots,N_{conjunctions}\}, y_{c_i} > 0\} \tag{4.18}$$

$$\varphi_\sigma(\mathbf{y}_{d_i}) = \{\sigma(Y_{d_i}) \mid i \in \{1,\dots,N_{disjunctions}\}, y_{d_i} > 0\} \tag{4.19}$$

4.2.2 Neural-to-Bivalent-Logic Translation with Truth Value Equivalence

This section demonstrates how to translate a discretised semi-symbolic node to an ASP program with the truth value equivalence. The focus is on solving the combined program of the rule translated from a discretised semi-symbolic node with weights $\mathbf{w} \in \{-6, 0, 6\}^N$, and the facts translated from an input $\mathbf{x} \in \{-1, 1\}^N$. At a high level, the translation of a semi-symbolic node is done by taking any positive weight and including the corresponding input element in the rule body; and any negative weight and including the corresponding input element with either classical negation or negation as failure (NAF) in the rule body. Similarly, any positive input element is mapped to a fact; and any negative input element is either mapped to a fact with classical negation or not included as part of the program. The inference result of the rules is the same regardless of whether the classical negation or NAF translation is chosen. This thesis chooses the NAF style of translation based on the projection function defined in Definition 4.2.2, so that if a fact is not present, it is considered as false.

The following defines the ASP translation of a discretised semi-symbolic node and proves that the soft-valued truth table of the node is the same as the truth table of the rule head.

Conjunctive Node

Definition 4.2.3 (ASP Translation of Discretised Conjunctive Node)**.** *Consider a discretised conjunctive node with N inputs, weights $\mathbf{w}_c$, and $\delta = 1$. The node should have at least one non-zero weight element, i.e. $|\mathcal{J}_{\mathbf{w}_c}| > 0$. This discretised conjunctive node is translated to an ASP rule in the form:*

$$c \leftarrow \bigwedge_{i \in \mathcal{J}^+_{\mathbf{w}_c}} a_i, \bigwedge_{i \in \mathcal{J}^-_{\mathbf{w}_c}} (\texttt{not } a_i) \tag{4.20}$$

where $\mathcal{J}^+_{\mathbf{w}_c}$ is the positive-sign relevant set and $\mathcal{J}^-_{\mathbf{w}_c}$ is the negative-sign relevant set defined in Definition 4.1.6. The input $\mathbf{x} \in \{-1, 1\}^N$ is translated to a set of facts using the projection function defined in Definition 4.2.2:

$$\begin{aligned} \mathcal{P}_{\mathbf{x},facts} &= \varphi_\sigma(\mathbf{x}) \\ &= \{a_i \mid i \in \{1,\dots,N\}, x_i = 1\} \end{aligned} \tag{4.21}$$

Theorem 4.2.1. *Given a discretised conjunctive semi-symbolic node with weights $\mathbf{w}_c \in \{-6, 0, 6\}^N$ and its translated ASP rule with rule head c based on the translation defined in Equation (4.20), and an input $\mathbf{x} \in \{-1, 1\}^N$ and its translated facts based on the translation defined in Equation (4.21), the bivalent logical interpretation b_c of the conjunctive semi-symbolic node from the input $\mathbf{x}$ is the same as the truth value of the rule head c evaluated with the joint ASP program of the translated rule and facts.*

Before the proof sketch and the full proof in Appendix A.2, an extension to the sign-based index notation is introduced. The argument requires reasoning about the interaction between the signs of the weight vector and the signs of the input vector. To facilitate this, a compact notation for representing the intersection of two sign-based index sets is defined.

Definition 4.2.4 (Intersection of Sign-Based Index Sets). *Recall the definition of sign-based index notation as defined in Definition 4.1.5. Consider two sign-based index sets $\mathcal{J}_{\mathbf{a}}^{op_1}$ and $\mathcal{J}_{\mathbf{b}}^{op_2}$ that both operate on the same index set $\mathcal{J}$. The intersection of these two sign-based index sets is represented as $\mathcal{J}_{\mathbf{a},\mathbf{b}}^{op_1,op_2}$. That is:*

$$\begin{aligned}\mathcal{J}_{\mathbf{a},\mathbf{b}}^{op_1,op_2} &= \mathcal{J}_{\mathbf{a}}^{op_1} \cap \mathcal{J}_{\mathbf{b}}^{op_2} \\ &= \{i \mid i \in \mathcal{J}, sign(a_i) = op_1, sign(b_i) = op_2\}\end{aligned} \tag{4.22}$$

where $op_1, op_2 \in \{+, -\}$.

Proof sketch of Theorem 4.2.1. The proof proceeds in three steps: (1) characterise the raw output in terms of sign agreement between $\mathbf{w}_c$ and $\mathbf{x}$; (2) relate the bivalent interpretation to the sign of that output using discretisation; and (3) show that the translated ASP rule body is satisfied under the translated facts if and only if the same sign conditions hold.

Step 1 (algebraic characterisation). Split $\mathcal{J}_{\mathbf{w}_c}$ into the four mutually exclusive subsets $\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{+,+}$, $\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{+,-}$, $\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{-,-}$, and $\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{-,+}$ using Definition 4.2.4. For discretised weights, the raw output simplifies to

$$f_{\mathbf{w}_c}(\mathbf{x}) = 6\left(1 - 2|\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{+,-}| - 2|\mathcal{J}_{\mathbf{w}_c,\mathbf{x}}^{-,+}|\right) \tag{4.23}$$

so only sign-mismatch indices affect the value.

Step 2 (neural bivalent interpretation). By Proposition 4.1.1, $f_{\mathbf{w}_c}(\mathbf{x})$ is either 6 or at most -6, hence $b_c = \top$ when $f_{\mathbf{w}_c}(\mathbf{x}) = 6$ and $b_c = \bot$ otherwise.

Step 3 (ASP equivalence). Facts $\mathcal{P}_{\mathbf{x},\text{facts}}$ contain a_i exactly when $x_i = 1$. The rule body in Equation (4.20) requires every positive-weight index to appear as a fact and every negative-weight index not to appear. Equation (4.23) gives $f_{\mathbf{w}_c}(\mathbf{x}) = 6$ if and only if both mismatch sets are empty, in which case the body is satisfied and c is in the answer set. If at least one mismatch index exists, then either some required fact is missing or some `not` a_i is false, the body fails, and c is not in the answer set. A complete proof, including the subcases for the latter situation, is given in Appendix A.2. ■

Disjunctive Node

Definition 4.2.5 (ASP Translation of Discretised Disjunctive Node). *Consider a discretised disjunctive node with N inputs (potentially from a conjunctive layer), weights $\mathbf{w}_d$, and $\delta = -1$. The node should have at least one non-zero weight element, i.e. $|\mathcal{J}_{\mathbf{w}_d}| > 0$. This discretised disjunctive node is translated to a set of ASP rules with the same rule head d:*

$$\mathcal{P}_{\mathbf{w}_d,rules} = \left\{d \leftarrow a_i. \mid i \in \mathcal{J}_{\mathbf{w}_d}^{+}\right\} \cup \left\{d \leftarrow \texttt{not } a_i. \mid i \in \mathcal{J}_{\mathbf{w}_d}^{-}\right\} \tag{4.24}$$

where $\mathcal{J}^{+}_{\mathbf{w}_d}$ is the positive-sign relevant set and $\mathcal{J}^{-}_{\mathbf{w}_d}$ is the negative-sign relevant set defined in Definition 4.1.6. The input $\mathbf{x} \in \{-1,1\}^N$ is translated to a set of facts using the projection function defined in Definition 4.2.2:

$$\begin{aligned} \mathcal{P}_{\mathbf{x},facts} &= \varphi_\sigma(\mathbf{x}) \\ &= \{a_i \mid i \in \{1,\dots,N\}, x_i = 1\} \end{aligned} \tag{4.25}$$

Note that the literals will be c_i instead of a_i in both $\mathcal{P}_{\mathbf{w}_d,rules}$ and $\mathcal{P}_{\mathbf{x},facts}$ if the inputs to the disjunctive node are outputs from a conjunctive layer.

Theorem 4.2.2. *Given a discretised disjunctive semi-symbolic node with weights $\mathbf{w}_d \in \{-6,0,6\}^N$ and its translated ASP rules with rule head d based on the translation defined in Equation (4.24), and an input $\mathbf{x} \in \{-1,1\}^N$ and its translated facts based on the translation defined in Equation (4.25), the bivalent logical interpretation b_d of the disjunctive semi-symbolic node from the input $\mathbf{x}$ is the same as the truth value of the rule head d evaluated with the joint ASP program of the translated rules and facts.*

Proof sketch of Theorem 4.2.2. The same sign-index decomposition as for Theorem 4.2.1 applies, but with the disjunctive bias from Equation (4.2) ($\delta = -1$). For discretised weights, the raw output satisfies

$$f_{\mathbf{w}_d}(\mathbf{x}) = 6\left(2|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| - 1\right) \tag{4.26}$$

By Proposition 4.1.2, $b_\mathrm{d} = \top$ when $f_{\mathbf{w}_d}(\mathbf{x}) \geq 6$ and $b_\mathrm{d} = \bot$ when $f_{\mathbf{w}_d}(\mathbf{x}) = -6$. The translated program in Equation (4.24) derives d when at least one rule body holds: either some $i \in \mathcal{J}^{+}_{\mathbf{w}_d}$ has a_i among the facts, or some $i \in \mathcal{J}^{-}_{\mathbf{w}_d}$ has a_i absent. Equation (4.26) shows that $f_{\mathbf{w}_d}(\mathbf{x}) \geq 6$ if and only if at least one sign-matching index exists in $\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}} \cup \mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}$, which is exactly when some translated rule body is true. When both matching sets are empty, every rule body fails and d is not in the answer set, matching $f_{\mathbf{w}_d}(\mathbf{x}) = -6$. A complete proof is given in Appendix A.2. ■

Translation of Discretised Neural DNF Model

Having established the translation of individual discretised conjunctive and disjunctive nodes to ASP rules, this section presents the translation of a complete discretised neural DNF model to an ASP program. A neural DNF model consists of a layer of conjunctive nodes followed by a layer of disjunctive nodes. The translation of the full model naturally composes the translations of its constituent layers, yielding an ASP program whose inference results correspond to the bivalent interpretations of the neural model.

Definition 4.2.6 (ASP Translation of Discretised Neural DNF Model)**.** *Consider a discretised neural DNF model with:*

- *N_{input} input variables;*
- *N_{conj} conjunctive nodes in the conjunctive layer, where the j-th conjunctive node has a weight vector $\mathbf{w}_{c_j} \in \{-6,0,6\}^{N_{input}}$;*
- *N_{disj} disjunctive nodes in the disjunctive layer, where the k-th disjunctive node has a weight vector $\mathbf{w}_{d_k} \in \{-6,0,6\}^{N_{conj}}$.*

The complete ASP program $\mathcal{P}_{NDNF}$ translated from this neural DNF model consists of two sets of rules:

$$\mathcal{P}_{NDNF} = \mathcal{P}_{conj} \cup \mathcal{P}_{disj} \tag{4.27}$$

where $\mathcal{P}_{conj}$ is the set of rules translated from the conjunctive layer and $\mathcal{P}_{disj}$ is the set of rules translated from the disjunctive layer.

Conjunctive Layer Translation. *Each conjunctive node c_j with at least one non-zero weight (i.e. $|\mathcal{J}_{\mathbf{w}_{c_j}}| > 0$) is translated to an ASP rule following Definition 4.2.3:*

$$\mathcal{P}_{conj} = \left\{ c_j \leftarrow \bigwedge_{i \in \mathcal{J}^+_{\mathbf{w}_{c_j}}} a_i, \bigwedge_{i \in \mathcal{J}^-_{\mathbf{w}_{c_j}}} (\texttt{not } a_i) \;\middle|\; j \in \{1, \ldots, N_{conj}\}, |\mathcal{J}_{\mathbf{w}_{c_j}}| > 0 \right\} \tag{4.28}$$

where a_i denotes the i-th input literal and c_j denotes the rule head corresponding to the j-th conjunctive node.

Disjunctive Layer Translation. *Each disjunctive node d_k with at least one non-zero weight (i.e. $|\mathcal{J}_{\mathbf{w}_{d_k}}| > 0$) is translated to a set of ASP rules following Definition 4.2.5:*

$$\mathcal{P}_{disj} = \bigcup_{\substack{k \in \{1, \ldots, N_{disj}\} \\ |\mathcal{J}_{\mathbf{w}_{d_k}}| > 0}} \left(\left\{ d_k \leftarrow c_j \mid j \in \mathcal{J}^+_{\mathbf{w}_{d_k}} \right\} \cup \left\{ d_k \leftarrow \texttt{not } c_j \mid j \in \mathcal{J}^-_{\mathbf{w}_{d_k}} \right\} \right) \tag{4.29}$$

where c_j denotes the j-th conjunction literal (corresponding to the output of the j-th conjunctive node) and d_k denotes the rule head corresponding to the k-th disjunctive node.

Input Translation. *Given an input vector $\mathbf{x} \in \{-1, 1\}^{N_{input}}$, the set of facts is generated using the projection function from Definition 4.2.2:*

$$\mathcal{P}_{\mathbf{x},facts} = \varphi_\sigma(\mathbf{x}) = \{ a_i \mid i \in \{1, \ldots, N_{input}\}, x_i = 1 \} \tag{4.30}$$

The combined ASP program $\mathcal{P}_{\mathbf{x},\text{facts}} \cup \mathcal{P}_{\text{NDNF}}$ forms a stratified logic program, where the conjunctive layer rules depend only on input facts and the disjunctive layer rules depend only on conjunctive layer atoms. This stratification ensures that the program has a unique answer set, which can be computed in two stages: first deriving the truth values of the conjunctive atoms from the input facts, then deriving the truth values of the disjunctive atoms from the conjunctive atoms.

Corollary 4.2.1 (Truth Value Equivalence of Neural DNF Model Translation). *Given a discretised neural DNF model and its translated ASP program $\mathcal{P}_{NDNF}$ as defined in Definition 4.2.6, and an input $\mathbf{x} \in \{-1, 1\}^{N_{input}}$ with its translated facts $\mathcal{P}_{\mathbf{x},facts}$, the following holds for all disjunctive nodes d_k with $|\mathcal{J}_{\mathbf{w}_{d_k}}| > 0$:*

$$b_{d_k} = \top \iff d_k \in AS(\mathcal{P}_{\mathbf{x},facts} \cup \mathcal{P}_{NDNF}) \tag{4.31}$$

where b_{d_k} is the bivalent logical interpretation of the k-th disjunctive node and $AS(\cdot)$ denotes the answer set of the program.

This corollary follows directly from the compositional application of Theorem 4.2.1 and Theorem 4.2.2. The conjunctive layer translation preserves truth values between the neural activations and the derived conjunctive atoms, and the disjunctive layer translation preserves truth values between the neural activations and the derived disjunctive atoms given the intermediate conjunctive layer outputs. Since the disjunctive layer takes the conjunctive layer's outputs as inputs, the composition of these two translations preserves the overall truth value equivalence from inputs to outputs.

Example 4.2.1: Translation of a Discretised Neural DNF Model

Consider a discretised neural DNF model for a multi-label classification task with 5 input variables, 4 conjunctive nodes, and 3 disjunctive nodes (representing three labels). The weight matrices are:

$$\mathbf{W}_{\text{conj}} = \begin{bmatrix} 6 & 0 & 0 & 0 & 0 \\ 0 & -6 & 0 & -6 & 0 \\ 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 6 & 6 \end{bmatrix} \qquad \mathbf{W}_{\text{disj}} = \begin{bmatrix} 6 & 0 & 0 & 0 \\ 0 & -6 & 6 & 0 \\ 0 & 0 & 0 & -6 \end{bmatrix}$$

where row j of $\mathbf{W}_{\text{conj}}$ corresponds to $\mathbf{w}_{\text{c}_j}$ and row k of $\mathbf{W}_{\text{disj}}$ corresponds to $\mathbf{w}_{\text{d}_k}$. And the biases are:

$$\beta_{\text{conj}} = \begin{bmatrix} 0 \\ -6 \\ 0 \\ -6 \end{bmatrix} \qquad \beta_{\text{disj}} = \begin{bmatrix} 0 \\ 6 \\ 0 \end{bmatrix}$$

For input $\mathbf{x} = [1, 1, 1, -1, -1]$, the output of the neural DNF model is:

$$\mathbf{c} = \tanh(\mathbf{W}_{\text{conj}}\mathbf{x} + \beta_{\text{conj}}) = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\mathbf{d} = \tanh(\mathbf{W}_{\text{disj}}\mathbf{c} + \beta_{\text{disj}}) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

The bivalent interpretations of the activations of the conjunctive and disjunctive nodes in the neural DNF model are:

$$[b_{\text{c}_1}, b_{\text{c}_2}, b_{\text{c}_3}, b_{\text{c}_4}] = [\top, \bot, \bot, \bot]$$
$$[b_{\text{d}_1}, b_{\text{d}_2}, b_{\text{d}_3}] = [\top, \top, \top]$$

The translated ASP program of this neural DNF model is $\mathcal{P}_{\text{NDNF}} = \mathcal{P}_{\text{conj}} \cup \mathcal{P}_{\text{disj}}$, where:

$$\mathcal{P}_{\text{conj}} = \left\{ \begin{array}{l} \mathrm{c}_1 \leftarrow \mathrm{a}_1. \\ \mathrm{c}_2 \leftarrow \texttt{not } \mathrm{a}_2, \texttt{not } \mathrm{a}_4. \\ \mathrm{c}_3 \leftarrow \texttt{not } \mathrm{a}_3. \\ \mathrm{c}_4 \leftarrow \mathrm{a}_4, \mathrm{a}_5. \end{array} \right\}$$

$$\mathcal{P}_{\text{disj}} = \left\{ \begin{array}{l} \mathrm{d}_1 \leftarrow \mathrm{c}_1. \\ \mathrm{d}_2 \leftarrow \texttt{not } \mathrm{c}_2. \\ \mathrm{d}_2 \leftarrow \mathrm{c}_3. \\ \mathrm{d}_3 \leftarrow \texttt{not } \mathrm{c}_4. \end{array} \right\}$$

For input $\mathbf{x} = [1, 1, 1, -1, -1]$, the facts are $\mathcal{P}_{\mathbf{x},\text{facts}} = \{\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3\}$. Computing the answer set of the combined program $\mathcal{P}_{\mathbf{x},\text{facts}} \cup \mathcal{P}_{\text{NDNF}}$ gives:

$$AS(\mathcal{P}_{\mathbf{x},\text{facts}} \cup \mathcal{P}_{\text{NDNF}}) = \{\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, \mathrm{c}_1, \mathrm{d}_1, \mathrm{d}_2, \mathrm{d}_3\}$$

The nodes with true bivalent interpretations (b_{c_1}, b_{d_1}, b_{d_2}, b_{d_3}) correspond exactly to the derived atoms in the answer set (c_1, d_1, d_2, d_3), confirming the truth value equivalence between the neural DNF model and its translated ASP program.

Algorithm 3 Neural DNF to ASP Translation

Input: Discretised neural DNF model with conjunctive weight matrix $\mathbf{W}_{\text{conj}} \in \{-6, 0, 6\}^{N_{\text{conj}} \times N_{\text{input}}}$, disjunctive weight matrix $\mathbf{W}_{\text{disj}} \in \{-6, 0, 6\}^{N_{\text{disj}} \times N_{\text{conj}}}$, and input vector $\mathbf{x} \in \{-1, 1\}^{N_{\text{input}}}$

Output: ASP program $\mathcal{P}_{\text{NDNF}}$ and set of facts $\mathcal{P}_{\mathbf{x},\text{facts}}$

procedure TRANSLATECONJLAYER($\mathbf{W}_{\text{conj}}$)
  $\mathcal{P}_{\text{conj}} \leftarrow \emptyset$
  for $j \in \{1, \ldots, N_{\text{conj}}\}$ **do**
    $\mathcal{J}^{+}_{\mathbf{w}_{\mathrm{c}_j}} \leftarrow \{i \mid w_{\mathrm{c}_j,i} = 6\}, \quad \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{c}_j}} \leftarrow \{i \mid w_{\mathrm{c}_j,i} = -6\}$
    if $|\mathcal{J}^{+}_{\mathbf{w}_{\mathrm{c}_j}} \cup \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{c}_j}}| > 0$ **then**
      Add rule $\mathrm{c}_j \leftarrow \bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_{\mathrm{c}_j}}} \mathrm{a}_i, \bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{c}_j}}} (\texttt{not}\ \mathrm{a}_i)$ to $\mathcal{P}_{\text{conj}}$
  return $\mathcal{P}_{\text{conj}}$
procedure TRANSLATEDISJLAYER($\mathbf{W}_{\text{disj}}$)
  $\mathcal{P}_{\text{disj}} \leftarrow \emptyset$
  for $k \in \{1, \ldots, N_{\text{disj}}\}$ **do**
    $\mathcal{J}^{+}_{\mathbf{w}_{\mathrm{d}_k}} \leftarrow \{j \mid w_{\mathrm{d}_k,j} = 6\}, \quad \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{d}_k}} \leftarrow \{j \mid w_{\mathrm{d}_k,j} = -6\}$
    if $|\mathcal{J}^{+}_{\mathbf{w}_{\mathrm{d}_k}} \cup \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{d}_k}}| > 0$ **then**
      for $j \in \mathcal{J}^{+}_{\mathbf{w}_{\mathrm{d}_k}}$ **do**
        Add rule $\mathrm{d}_k \leftarrow \mathrm{c}_j$ to $\mathcal{P}_{\text{disj}}$
      for $j \in \mathcal{J}^{-}_{\mathbf{w}_{\mathrm{d}_k}}$ **do**
        Add rule $\mathrm{d}_k \leftarrow \texttt{not}\ \mathrm{c}_j$ to $\mathcal{P}_{\text{disj}}$
  return $\mathcal{P}_{\text{disj}}$
$\mathcal{P}_{\text{conj}} \leftarrow$ TRANSLATECONJLAYER($\mathbf{W}_{\text{conj}}$)
$\mathcal{P}_{\text{disj}} \leftarrow$ TRANSLATEDISJLAYER($\mathbf{W}_{\text{disj}}$)
$\mathcal{P}_{\text{NDNF}} \leftarrow \mathcal{P}_{\text{conj}} \cup \mathcal{P}_{\text{disj}}$
$\mathcal{P}_{\mathbf{x},\text{facts}} \leftarrow \{\mathrm{a}_i \mid i \in \{1, \ldots, N_{\text{input}}\},\ x_i = 1\}$ ▷ Translate input vector to facts
return $\mathcal{P}_{\text{NDNF}},\ \mathcal{P}_{\mathbf{x},\text{facts}}$

4.2.3 Bivalent-Logic-to-Neural Translation with Truth Value Equivalence

This section demonstrates how to translate a **stratified** ASP program to a discretised neural DNF model with truth value equivalence. The translation is developed incrementally: the first two subsections establish translations for individual rule forms (a single conjunction rule, and a group of disjunction rules sharing the same head), before the third subsection combines these into a full translation for a complete stratified ASP program (Definition 4.2.9). Stratification is the key structural assumption that makes the compositional translation correct: it ensures a well-defined two-level evaluation order (input facts → conjunction rules → disjunction rules) with no circular dependencies. Note that constraints, aggregations, or choice rules cannot be translated into a discretised semi-symbolic node. Furthermore, if the ASP program is first-order, it must be grounded before translation.

Pure Conjunction Rule

Definition 4.2.7 (Translation of Single Rule to Discretised Conjunctive Semi-Symbolic Node)**.** *There are N different possible facts $a_1, a_2, \ldots, a_N$. Consider an ASP rule with head h_c in the form:*

$$h_c \leftarrow \bigwedge_{i \in \mathcal{J}^+_{\mathbf{w}_c}} a_i, \bigwedge_{i \in \mathcal{J}^-_{\mathbf{w}_c}} (\texttt{not } a_i) \tag{4.32}$$

where $\mathcal{J}^+_{\mathbf{w}_c}$ is the set of indices of the positive literals in the rule, and $\mathcal{J}^-_{\mathbf{w}_c}$ is the set of indices of the negated literals in the rule. The following conditions hold:

$$\mathcal{J}^+_{\mathbf{w}_c} \subseteq \{1, \ldots, N\}, \mathcal{J}^-_{\mathbf{w}_c} \subseteq \{1, \ldots, N\}, |\mathcal{J}^+_{\mathbf{w}_c}| \geq 0, |\mathcal{J}^-_{\mathbf{w}_c}| \geq 0 \tag{4.33}$$

$$\mathcal{J}^+_{\mathbf{w}_c} \cap \mathcal{J}^-_{\mathbf{w}_c} = \emptyset \tag{4.34}$$

The relevant set is defined as $\mathcal{J}_{\mathbf{w}_c} = \mathcal{J}^+_{\mathbf{w}_c} \cup \mathcal{J}^-_{\mathbf{w}_c}$, and $|\mathcal{J}_{\mathbf{w}_c}| \neq 0$.

Note that h_c cannot appear in the body of the rule, that is, $h_c \neq a_i$ for any $i \in \{1, \ldots, N\}$.

*This rule can be translated to a **conjunctive** semi-symbolic node with N inputs and $\delta = 1$, and its weight vector $\mathbf{w}_c \in \{-6, 0, 6\}^N$, where:*

$$w_{c,i} = \begin{cases} 6 & \text{if } i \in \mathcal{J}^+_{\mathbf{w}_c} \\ -6 & \text{if } i \in \mathcal{J}^-_{\mathbf{w}_c} \\ 0 & \text{otherwise} \end{cases} \tag{4.35}$$

A set of facts $\mathcal{P}_{\mathbf{x},facts} \subseteq \{a_i \mid i \in \{1, \ldots, N\}\}$ is translated to an input $\mathbf{x} \in \{-1, 1\}^N$ as:

$$\forall i \in \{1, \ldots, N\}.\ \ x_i = \begin{cases} 1 & \text{if } a_i \in \mathcal{P}_{\mathbf{x},facts} \\ -1 & \text{otherwise} \end{cases} \tag{4.36}$$

Theorem 4.2.3. *Given an ASP rule in the form of (4.32) with head h_c that satisfies the conditions defined in Equation (4.33) and (4.34) and its translated conjunctive semi-symbolic node with weights based on the translation defined in Equation (4.35), and the set of facts ASP program and its translated input vector based on the translation defined in Equation (4.36), the truth value of h_c evaluated*

with the joint ASP program of the rule and facts is the same as the bivalent logical interpretation b_c of the translated conjunctive semi-symbolic node computed from the translated input vector.

Proof sketch of Theorem 4.2.3. The translated node uses the same weights and input encoding as the neural-to-logic direction, so Equation (4.23) still characterises $f_{\mathbf{w}_c}(\mathbf{x})$. If the rule head h_c is in the answer set, its body requires $x_i = 1$ for all $i \in \mathcal{J}^+_{\mathbf{w}_c}$ and $x_i = -1$ for all $i \in \mathcal{J}^-_{\mathbf{w}_c}$, hence both mismatch sets are empty and $f_{\mathbf{w}_c}(\mathbf{x}) = 6$, so $b_c = \top$. If h_c is not in the answer set, at least one positive literal is false or one NAF literal is false under the facts, which places at least one index in $\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}$ or $\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}$, giving $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6$ and $b_c = \bot$. A complete proof is given in Appendix A.2. ■

Multiple Rules with the Same Rule Head

Definition 4.2.8 (Translation of Multiple Rules to Discretised Disjunctive Semi-Symbolic Node)**.** *Consider multiple ASP rules with the same rule head h_d, with N different possible facts $a_1, a_2, \ldots, a_N$ in the body. The set of rules is in the form of:*

$$\mathcal{P}_{\mathbf{w}_d,rules} = \left\{h_d \leftarrow a_i. \mid i \in \mathcal{J}^+_{\mathbf{w}_d}\right\} \cup \left\{h_d \leftarrow \texttt{not } a_i. \mid i \in \mathcal{J}^-_{\mathbf{w}_d}\right\} \tag{4.37}$$

where $\mathcal{J}^+_{\mathbf{w}_d}$ is the set of indices of the positive literals, and $\mathcal{J}^-_{\mathbf{w}_d}$ is the set of indices of the negated literals. The following conditions hold:

$$\mathcal{J}^+_{\mathbf{w}_d} \subseteq \{1,\ldots,N\}, \mathcal{J}^-_{\mathbf{w}_d} \subseteq \{1,\ldots,N\}, |\mathcal{J}^+_{\mathbf{w}_d}| \geq 0, |\mathcal{J}^-_{\mathbf{w}_d}| \geq 0 \tag{4.38}$$

$$\mathcal{J}^+_{\mathbf{w}_d} \cap \mathcal{J}^-_{\mathbf{w}_d} = \emptyset \tag{4.39}$$

The relevant set is defined as $\mathcal{J}_{\mathbf{w}_d} = \mathcal{J}^+_{\mathbf{w}_d} \cup \mathcal{J}^-_{\mathbf{w}_d}$, and $|\mathcal{J}_{\mathbf{w}_d}| \neq 0$.

Note that h_d cannot appear in the body of the rules, that is, $h_d \neq a_i$ for any $i \in \{1,\ldots,N\}$.

*This program can be translated to a **disjunctive** semi-symbolic node with N inputs and $\delta = -1$, and its weight vector $\mathbf{w}_d \in \{-6, 0, 6\}^N$, where:*

$$w_{d,i} = \begin{cases} 6 & \text{if } i \in \mathcal{J}^+_{\mathbf{w}_d} \\ -6 & \text{if } i \in \mathcal{J}^-_{\mathbf{w}_d} \\ 0 & \text{otherwise} \end{cases} \tag{4.40}$$

A set of facts $\mathcal{P}_{\mathbf{x},facts} \subseteq \{a_i \mid i \in \{1,\ldots,N\}\}$ is translated to an input vector $\mathbf{x} \in \{-1, 1\}^N$ as:

$$\forall i \in \{1,\ldots,N\}.\ \ x_i = \begin{cases} 1 & \text{if } a_i \in \mathcal{P}_{\mathbf{x},facts} \\ -1 & \text{otherwise} \end{cases} \tag{4.41}$$

Note that the rules in $\mathcal{P}_{\mathbf{w}_d,\text{rules}}$ all have only one body literal. For rules that have more than one body literal, an auxiliary predicate can be invented to represent the conjunction of the body literals beforehand to match the form.

Theorem 4.2.4. *Given some ASP rules with the same rule head h_d in the form of (4.37) that satisfies the conditions defined in Equation (4.38) and (4.39) and its translated disjunctive semi-symbolic node with weights based on the translation defined in Equation (4.40), and the set of facts ASP program and its translated input vector based on the translation defined in Equation (4.41),*

the truth value of h_d evaluated with the joint ASP program of the rules and facts is the same as the bivalent logical interpretation b_d of the translated disjunctive semi-symbolic node computed from the translated input vector.

Proof sketch of Theorem 4.2.4. As above, Equation (4.26) applies to the translated disjunctive node. Head h_d is in the answer set when at least one rule body is satisfied, which forces at least one index in $\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}$ or $\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}$, hence $f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) \geq 6$ and $b_\mathrm{d} = \top$. If no body is satisfied, every $i \in \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d}}$ has $x_i = -1$ and every $i \in \mathcal{J}^{-}_{\mathbf{w}_\mathrm{d}}$ has $x_i = 1$, both matching sets are empty, $f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) = -6$, and $b_\mathrm{d} = \bot$. A complete proof is given in Appendix A.2. ■

Translation of Stratified ASP Program

Having established the translation of individual ASP rules to conjunctive and disjunctive semi-symbolic nodes, this section presents the translation of a complete stratified ASP program to a discretised neural DNF model. The ASP program must satisfy certain structural constraints to be translatable: it must be stratified with two levels (conjunction rules followed by disjunction rules), and it cannot contain constraints, aggregations, choice rules, or recursive definitions.

Definition 4.2.9 (Translatable Stratified ASP Program). *A **translatable stratified ASP program** $\mathcal{P}$ consists of:*

- *A set of N_{input} possible input facts $\{a_1, \ldots, a_{N_{input}}\}$;*
- *A set of N_{conj} conjunction rules $\mathcal{P}_{conj}$, where each rule $j \in \{1, \ldots, N_{conj}\}$ has a distinct head c_j and is in the form of Equation (4.32):*

$$c_j \leftarrow \bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_{c_j}}} a_i, \bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_{c_j}}} (\texttt{not } a_i)$$

  where the body contains only input literals a_i;

- *A set of N_{disj} groups of disjunction rules $\mathcal{P}_{disj}$, where each group $k \in \{1, \ldots, N_{disj}\}$ shares the same head d_k and is in the form of Equation (4.37):*

$$\mathcal{P}_{\mathbf{w}_{d_k}, rules} = \left\{ d_k \leftarrow c_j \mid j \in \mathcal{J}^{+}_{\mathbf{w}_{d_k}} \right\} \cup \left\{ d_k \leftarrow \texttt{not } c_j \mid j \in \mathcal{J}^{-}_{\mathbf{w}_{d_k}} \right\}$$

  where the body contains only conjunction literals c_j.

The program is stratified: stratum 0 contains the input facts, stratum 1 contains the conjunction rules (whose bodies depend only on stratum 0), and stratum 2 contains the disjunction rules (whose bodies depend only on stratum 1).

Definition 4.2.10 (Neural DNF Translation of Stratified ASP Program). *Given a translatable stratified ASP program $\mathcal{P} = \mathcal{P}_{conj} \cup \mathcal{P}_{disj}$ as defined in Definition 4.2.9, the program is translated to a discretised neural DNF model as follows:*

Conjunctive Layer Translation. *Each conjunction rule with head c_j is translated to a conjunctive semi-symbolic node following Definition 4.2.7. The conjunctive layer has N_{conj} nodes, with each node j having weight vector $\mathbf{w}_{c_j} \in \{-6, 0, 6\}^{N_{input}}$ defined by Equation (4.35).*

Disjunctive Layer Translation. *Each group of disjunction rules with head d_k is translated to a disjunctive semi-symbolic node following Definition 4.2.8. The disjunctive layer has N_{disj} nodes, with each node k having weight vector $\mathbf{w}_{d_k} \in \{-6, 0, 6\}^{N_{conj}}$ defined by Equation (4.40).*

Input Translation. *A set of input facts $\mathcal{P}_{\mathbf{x},facts} \subseteq \{a_i \mid i \in \{1, \ldots, N_{input}\}\}$ is translated to an input vector $\mathbf{x} \in \{-1, 1\}^{N_{input}}$ following Equation (4.36).*

Corollary 4.2.2 (Truth Value Equivalence of Stratified ASP Translation)**.** *Given a translatable stratified ASP program $\mathcal{P}$ and its translated discretised neural DNF model as defined in Definition 4.2.10, and a set of input facts $\mathcal{P}_{\mathbf{x},facts}$ with its translated input vector $\mathbf{x}$, the following holds for all disjunction rule heads d_k:*

$$d_k \in AS(\mathcal{P}_{\mathbf{x},facts} \cup \mathcal{P}) \iff b_{d_k} = \top \tag{4.42}$$

where $AS(\cdot)$ denotes the answer set of the program and b_{d_k} is the bivalent logical interpretation of the k-th disjunctive node.

This corollary follows directly from the compositional application of Theorem 4.2.3 and Theorem 4.2.4. The stratification ensures that the conjunction rules are evaluated first using only the input facts, and the disjunction rules are evaluated second using only the derived conjunction atoms. Since the neural DNF model's forward pass computes the conjunctive layer outputs before the disjunctive layer outputs, the compositional application of the two theorems preserves the overall truth value equivalence.

Example 4.2.2: Translation of a Stratified ASP Program

Consider the following stratified ASP program with 4 input facts, 3 conjunction rules, and 2 groups of disjunction rules:

$$\mathcal{P}_{\text{conj}} = \left\{ \begin{array}{l} \mathrm{c}_1 \leftarrow \mathrm{a}_1, \mathrm{a}_2. \\ \mathrm{c}_2 \leftarrow \mathrm{a}_3, \texttt{not } \mathrm{a}_4. \\ \mathrm{c}_3 \leftarrow \texttt{not } \mathrm{a}_1, \texttt{not } \mathrm{a}_2. \end{array} \right\}$$

$$\mathcal{P}_{\text{disj}} = \left\{ \begin{array}{l} \mathrm{d}_1 \leftarrow \mathrm{c}_1. \\ \mathrm{d}_1 \leftarrow \mathrm{c}_2. \\ \mathrm{d}_2 \leftarrow \texttt{not } \mathrm{c}_1. \\ \mathrm{d}_2 \leftarrow \mathrm{c}_3. \end{array} \right\}$$

The translated neural DNF model has weight matrices:

$$\mathbf{W}_{\text{conj}} = \begin{bmatrix} 6 & 6 & 0 & 0 \\ 0 & 0 & 6 & -6 \\ -6 & -6 & 0 & 0 \end{bmatrix} \qquad \mathbf{W}_{\text{disj}} = \begin{bmatrix} 6 & 6 & 0 \\ -6 & 0 & 6 \end{bmatrix}$$

where row j of $\mathbf{W}_{\text{conj}}$ corresponds to $\mathbf{w}_{\mathrm{c}_j}$ and row k of $\mathbf{W}_{\text{disj}}$ corresponds to $\mathbf{w}_{\mathrm{d}_k}$. The biases are $\boldsymbol{\beta}_{\text{conj}} = [-6, -6, -6]^\top$ and $\boldsymbol{\beta}_{\text{disj}} = [6, 6]^\top$.
For input facts $\mathcal{P}_{\mathbf{x},\text{facts}} = \{\mathrm{a}_1, \mathrm{a}_3\}$, the translated input vector is $\mathbf{x} = [1, -1, 1, -1]$. Comput-

ing the answer set of the combined program yields:

$$\mathrm{AS}(\mathcal{P}_{\mathbf{x},\text{facts}} \cup \mathcal{P}) = \{\mathrm{a}_1, \mathrm{a}_3, \mathrm{c}_2, \mathrm{d}_1, \mathrm{d}_2\}$$

The neural DNF model's forward pass produces:

$$\mathbf{c} = \tanh(\mathbf{W}_{\text{conj}}\mathbf{x} + \boldsymbol{\beta}_{\text{conj}}) = [-1, 1, -1]^\top$$
$$\mathbf{d} = \tanh(\mathbf{W}_{\text{disj}}\mathbf{c} + \boldsymbol{\beta}_{\text{disj}}) = [1, 1]^\top$$

The bivalent interpretations are $[b_{\mathrm{c}_1}, b_{\mathrm{c}_2}, b_{\mathrm{c}_3}] = [\bot, \top, \bot]$ and $[b_{\mathrm{d}_1}, b_{\mathrm{d}_2}] = [\top, \top]$. The nodes with true bivalent interpretations (b_{c_2}, b_{d_1}, b_{d_2}) correspond exactly to the derived atoms in the answer set (c_2, d_1, d_2), confirming the truth value equivalence.

Algorithm 4 ASP to Neural DNF Translation

Input: Translatable stratified ASP program $\mathcal{P} = \mathcal{P}_{\text{conj}} \cup \mathcal{P}_{\text{disj}}$ with N_{input} possible input facts $\{\mathrm{a}_1, \ldots, \mathrm{a}_{N_{\text{input}}}\}$, N_{conj} conjunction rules, N_{disj} groups of disjunction rules, and set of input facts $\mathcal{P}_{\mathbf{x},\text{facts}}$

Output: Discretised neural DNF model weights $\mathbf{W}_{\text{conj}}$, $\mathbf{W}_{\text{disj}}$, and input vector $\mathbf{x}$

procedure TRANSLATECONJRULES($\mathcal{P}_{\text{conj}}$)
  $\mathbf{W}_{\text{conj}} \leftarrow \mathbf{0}^{N_{\text{conj}} \times N_{\text{input}}}$
  for each rule j with head c_j in $\mathcal{P}_{\text{conj}}$ **do**
    for each positive literal a_i in the rule body **do**
      $w_{\mathrm{c}_j,i} \leftarrow 6$
    for each negated literal **not** a_i in the rule body **do**
      $w_{\mathrm{c}_j,i} \leftarrow -6$
  return $\mathbf{W}_{\text{conj}}$
procedure TRANSLATEDISJRULES($\mathcal{P}_{\text{disj}}$)
  $\mathbf{W}_{\text{disj}} \leftarrow \mathbf{0}^{N_{\text{disj}} \times N_{\text{conj}}}$
  for each group k of rules with head d_k in $\mathcal{P}_{\text{disj}}$ **do**
    for each rule $\mathrm{d}_k \leftarrow \mathrm{c}_j$ in the group **do**
      $w_{\mathrm{d}_k,j} \leftarrow 6$
    for each rule $\mathrm{d}_k \leftarrow$ **not** c_j in the group **do**
      $w_{\mathrm{d}_k,j} \leftarrow -6$
  return $\mathbf{W}_{\text{disj}}$
$\mathbf{W}_{\text{conj}} \leftarrow$ TRANSLATECONJRULES($\mathcal{P}_{\text{conj}}$)
$\mathbf{W}_{\text{disj}} \leftarrow$ TRANSLATEDISJRULES($\mathcal{P}_{\text{disj}}$)
▷ Compute biases from the derived weight matrices
$\beta_{\mathrm{c}_j} \leftarrow 6 - 6|\mathcal{J}_{\mathbf{w}_{\mathrm{c}_j}}|$ for each $j \in \{1, \ldots, N_{\text{conj}}\}$
$\beta_{\mathrm{d}_k} \leftarrow 6|\mathcal{J}_{\mathbf{w}_{\mathrm{d}_k}}| - 6$ for each $k \in \{1, \ldots, N_{\text{disj}}\}$
$x_i \leftarrow 1$ if $\mathrm{a}_i \in \mathcal{P}_{\mathbf{x},\text{facts}}$, else $x_i \leftarrow -1$, for each $i \in \{1, \ldots, N_{\text{input}}\}$ ▷ Translate facts to input vector
return $\mathbf{W}_{\text{conj}}$, $\mathbf{W}_{\text{disj}}$, $\mathbf{x}$

Chapter 5

Neural DNF-based Models for Classification Tasks

The main criticism of the pix2rule [28] work is that the neural DNF model has only been empirically evaluated on binary and synthetic classification tasks. To further explore the capabilities of the neural DNF model, this chapter experiments on two non-binary real-world classification tasks, the *multi-class* dataset CUB-200-2011 [157] and its subsets, and the *multi-label* dataset TMC2007-500 [148] and its subsets. Furthermore, a novel variation of the neural DNF model called the neural DNF-EO model is proposed for multi-class classification tasks, to support logical mutually exclusive rule learning, i.e. ensuring exactly one class is true at a time. Mutual exclusivity can be enforced by logical constraints for a symbolic rule learner. However, cross-entropy loss does not strictly enforce such behaviour for a neural architecture, but rather pushes the output logits to be different from each other. While the neural DNF-EO model is trained with a cross-entropy loss, it enforces mutual exclusion in a differentiable fashion with a layer of frozen weights acting as a neuro-symbolic constraint.

This chapter focuses on exploring (i) the classification performance and scalability of the neural DNF-based models over an increasing number of discrete attributes, and (ii) the interpretability of the learned rules versus other pure symbolic and classical classification methods such as decision trees. The experimental results show that the neural DNF-based models perform at levels competitive with pure neural networks, whilst providing symbolic rules through post-training processes. They scale better than a state-of-the-art pure symbolic rule learner FastLAS [91] over large search spaces. The neural DNF-based models successfully solve all tasks while FastLAS fails to learn in multi-class classification tasks with 200 classes and in all multi-label settings. Compared to other differentiable ILP methods (e.g. [50, 43, 170]), the neural DNF-based models do not require human-engineered rule templates, provide interpretability by enabling the extraction of symbolic rules after training, and handle logical negation without special treatment.

The rest of this chapter includes material from our paper [15].

5.1 Neural DNF-EO Model

This section first discusses the challenge of multi-class classification and then introduces the neural DNF-EO model architecture, which is a neural DNF-based model that enforces mutual exclusivity on the model's outputs. Finally, the modified post-training process based on the pipeline in Section 2.3.3 is introduced.

5.1.1 The Challenge of Multi-class Classification

A multi-class classification task requires exactly one class to be true at a time. An MLP handles this naturally: with a softmax-activated output layer, its outputs are interpreted as class probabilities summing to 1, and cross-entropy loss encourages the logits to separate so that one class has a clearly larger value than the others. At inference time, the predicted class is obtained by taking the arg max of the logits without explicitly applying softmax.

In contrast, a neural DNF model encodes a set of *independent* rules. Each output's bivalent interpretation is determined by a threshold at 0 (True if greater than 0, False otherwise) and is independent of all other outputs, since the architecture contains no loops. Satisfying mutual exclusivity therefore requires the training process to push the output logits not only to be *different* from one another, but also to be *extreme* values lying on opposite sides of the threshold. Cross-entropy loss does not enforce this: the softmax function merely amplifies relative differences between logits and does not ensure that each logit crosses the threshold required for a consistent logical interpretation. This creates a tension between *sufficiency* (the rules must be sufficient to imply class j) and *necessity* (no class other than j must be implied by the rules), as illustrated in Example 5.1.1. The experiment in Section 5.2.1 empirically confirms this: when trained with cross-entropy loss, plain neural DNF models may predict multiple classes as true when taking the bivalent interpretation of the disjunctive layer's outputs.

Example 5.1.1: Cross-entropy loss does not enforce extreme logit values

Suppose the raw output logits of a neural DNF model are $\tilde{y} = [-2, 1.5, 3]$ with a target class of 2 (zero-indexed). Then $\text{softmax}(\tilde{y}) = [0.006, 0.181, 0.813]$, which exhibits clear separation and yields a cross-entropy loss of only 0.207. However, applying the tanh activation gives $\tanh(\tilde{y}) = [-0.964, 0.905, 0.995]$, which is interpreted as $[\bot, \top, \top]$. This is a violation of the 'exactly-one-class' property.

One could attempt to encode mutual exclusivity directly in the rules by designating a dominant class and requiring the rules for all other classes to additionally negate that class's conditions, as shown in Example 5.1.2. In practice, however, identifying a dominant class and determining the required conjunctions is difficult, and the model tends to struggle when learning conjunctions with highly similar conjuncts (explored further in Chapter 7 as the entanglement problem). Logic programs avoid this difficulty by expressing the 'exactly-one' property as a single integrity constraint: $\leftarrow class(X), class(Y), X \neq Y$. This motivates adding an extra layer to the neural DNF model to mimic such a constraint, yielding the neural DNF-EO architecture, which (i) can be trained with cross-entropy loss and (ii) nonetheless guarantees mutual exclusivity. The architecture of this extra layer and its interaction with the plain neural DNF are described in the following section.

Example 5.1.2: Dominant-class rule expansion for mutual exclusivity

If class k_1 is the dominant class with associated rule $d_{k_1} \leftarrow l_{k_1,1}, \ldots, l_{k_1,I_1}$, then the rule for class k_2 with original form $d_{k_2} \leftarrow l_{k_2,1}, \ldots, l_{k_2,I_2}$ becomes $d_{k_2} \leftarrow l_{k_2,1}, \ldots, l_{k_2,I_2}, \texttt{not}\ d_{k_1}$ to reflect its dependency on the dominant class k_1. This rule can be unfolded to remove the negative loop introduced by the negation:

$$
\begin{aligned}
d_{k_2} &\leftarrow l_{k_2,1}, \ldots l_{k_2,I_2}, \texttt{not}\ l_{k_1,1} \\
d_{k_2} &\leftarrow l_{k_2,1}, \ldots l_{k_2,I_2}, \texttt{not}\ l_{k_1,2} \\
&\vdots \\
d_{k_2} &\leftarrow l_{k_2,1}, \ldots l_{k_2,I_2}, \texttt{not}\ l_{k_1,I_1}
\end{aligned}
$$

Each rule body is a conjunction learnable by a conjunctive node, so a sufficiently wide neural DNF model could in principle learn mutually exclusive rules by adding negated atoms from the dominant class's rule body to every other class's rules.

5.1.2 Model Architecture

The **neural DNF-EO model** shown in Figure 5.1 has a constraint layer (layer C2 in the figure) that reflects the 'exactly-one' constraint '$class_i \leftarrow \wedge_{j,j\neq i}$ not $class_j$' which enforces mutual exclusion on the plain neural DNF (in the dotted yellow box in the figure). If more than one output is true after the disjunctive layer C1, no class would be true in the constraint layer C2. This constraint layer C2 is initialised with all connections set to -6[1], representing negation, and its weights are not updated. During training, the constraint layer C2's output logits are pushed to be different and to match the expected class by the cross-entropy loss, but the output of disjunctive layer C1 of the plain neural DNF is pushed to have only one value being true and all the rest being false. At inference time, the constraint layer C2 is removed and inference is performed with the plain neural DNF that is enforced to output exactly one class as true at a time, thus providing mutual exclusivity.

[1]Setting weights to 6 or -6 saturates tanh ($\tanh(\pm 6) \approx \pm 1$), so the output is approximately 1 or -1.

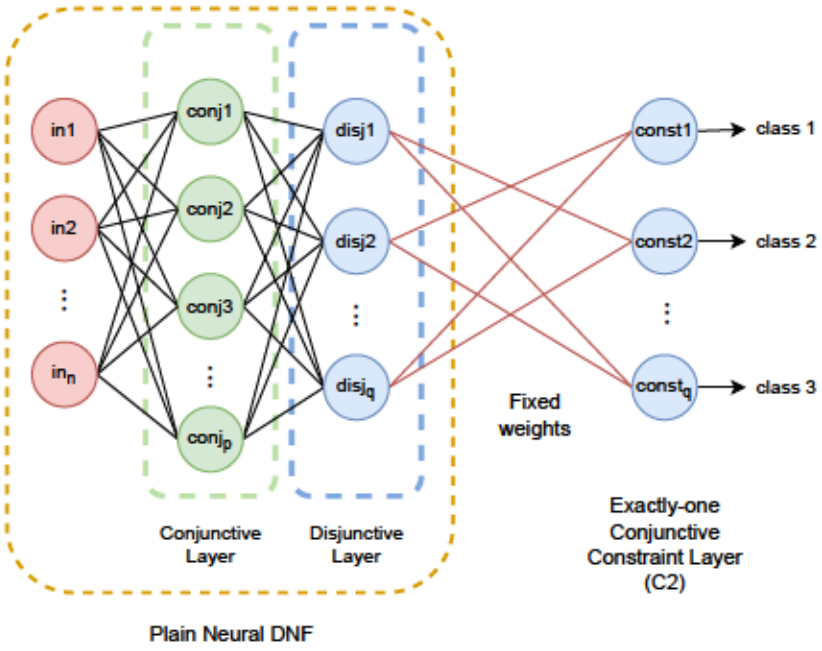


Figure 5.1: An example of a neural DNF-EO model.

5.1.3 Post-training Process

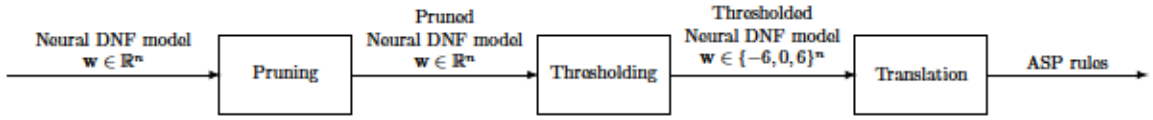
For this chapter's experiments, the post-training process in Section 2.3.3 is refined to incorporate the following four stages: pruning, finetuning, thresholding and extraction, as shown in Figure 5.2. The procedures of pruning, thresholding and extraction remain the same as in Section 2.3.3, while finetuning is added as a new step to adjust for learning and extracting rules in real-world datasets.

Pruning As described in Section 2.3.3, this step removes edges/weights that do not significantly degrade model performance. For the neural DNF-EO model, the constraint layer C2 is removed, and the pruning is only applied to the plain neural DNF. The performance metric chosen is the Jaccard index score.

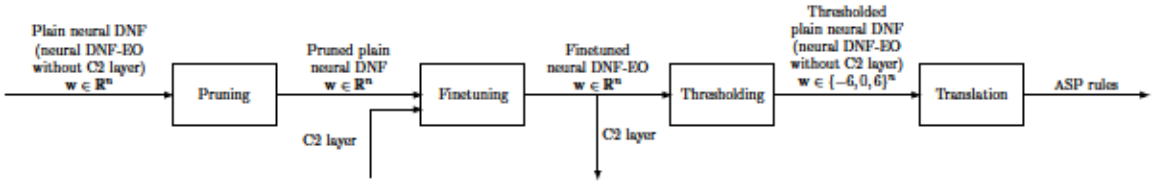
Finetuning This new stage adjusts the model's weights by re-training the pruned neural DNF-based model on the training set without updating the pruned weights. For the neural DNF-EO model, the constraint layer is added back after the pruned plain neural DNF during the finetuning process to maintain mutual exclusivity.

Thresholding Following the same procedure as in Section 2.3.3, this step converts the continuous weights from $\mathbb{R}$ to discrete values in $\{-6, 0, 6\}$ to prepare for logical translation. For the neural DNF-EO model, thresholding is applied to the finetuned plain neural DNF.

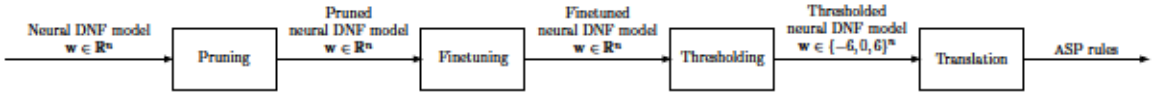
Extraction The final step translates the thresholded plain neural DNF model to ASP rules (Section 2.3.3), where each conjunctive node becomes a rule and weights in $\{-6, 6\}$ indicate the presence of atoms in the rule body. The bidirectional translation in Section 4.2 still applies here, since the logical translation is done on the two-layered neural DNF model after removing the exactly-one constraint.



(a) The post-training process to extract ASP rules from a neural DNF model. This is identical to Figure 2.5 in Section 2.3.3.



(b) The post-training process to extract ASP rules from a neural DNF-EO model. The constraint layer C2 is first removed before the pruning stage. It is then added back at the finetuning stage to combine with the pruned plain neural DNF to form the full neural DNF-EO model, in order to maintain mutual exclusivity. After finetuning, the constraint layer C2 is removed again and only the plain finetuned neural DNF is thresholded and translated to ASP rules.



(c) The post-training process to extract ASP rules from a neural DNF model. This is the same as the pipeline in (a), except for the addition of the finetuning step.

Figure 5.2: The post-training pipelines used in this chapter's experiments to extract ASP rules from a neural DNF-EO model (b) and from a neural DNF model (c), in comparison to the post-training process (a) in Section 2.3.3.

5.2 Experiments

Two types of neural DNF-based models are trained in this section: the (plain) neural DNF model and the neural DNF-EO model, and they are evaluated in terms of *classification performance*, *scalability*, and *interpretability*. The experiments are organised in three parts according to output structure and the dataset type. Part I (real-world multi-class classification) evaluates neural DNF-EO on mutually exclusive class outputs in Section 5.2.1, including whether cross-entropy training of a plain neural DNF suffices without the EO constraint layer. Part II (real-world multi-label classification) evaluates plain neural DNF on independently activating labels in Section 5.2.2, where mutual exclusivity is neither required nor imposed. This part also supports the claim that the neural DNF model itself is capable to learn independent rules but requires special adjustment for multi-class classification where the rules are mutually exclusive.[2] Part III (synthetic datasets) serves as a sanity check for the neural DNF-based models' capabilities of learning rules from datasets with ground truth logic programs and different scale of input/output. All experiments use the same propositional inputs, training, rule-extraction pipeline, and scalability-subset methodology. They differ in the loss function, output semantics, and the model variant under test. A two-layer MLP is selected as the neural baseline for comparison, since it also does not rely on background knowledge or mode biases to bootstrap, serving as a representative upper bound for neural classification performance without

[2]Chapter 7 also includes multi-label experiments on Boolean-network benchmarks with ground-truth logic programs. Those experiments target faithful rule extraction via disentanglement rather than the large-scale scalability comparison to FastLAS conducted here.

symbolic constraints. Since this chapter focuses on scalable and interpretable rule learning from propositions, models without rule-based interpretations are not meaningful competitors here,[3] and the MLP alone is sufficient to contextualise the raw predictive performance of the neural DNF-based models. For comparison with other learners with rule-based or rule-like interpretability, the baselines are the state-of-the-art symbolic learner FastLAS [91] and decision trees, which both provide rule-based interpretation. For FastLAS, the classification task is treated as a propositional ILP task with no background knowledge, and the scoring function is set to use the body length so that FastLAS learns the most compact rules.

5.2.1 Real-world Multi-class Task: CUB Dataset/Subsets

Dataset Description

CUB-200-2011 [157] is a real-world multi-class bird classification dataset with crowd-worker-annotated bird attributes as concepts. A sample such as in Figure 5.3 contains an image, a set of 312 attribute annotations and a class label out of 200 possible classes. Having attribute annotations in the dataset gives two viable experiment configurations. One is to train two separate models, one for attribute prediction and one for classification, and at inference time combine the two models to make the final prediction. The other option is to train an end-to-end model from images to classification and try to align the latent space to the ground truth attribute annotations, similar to Concept Bottleneck Models (CBMs) [88]. While the CBM work [88] suggests that CBMs provide both interpretability at the concept level and intervenability such that humans can alter the concepts before the model makes any final decision, it is argued that only the combination of an independently trained concepts model and an independently trained classification model can satisfy such interpretability and intervenability [103]. The work in [103] also observes that end-to-end CBMs may not use predicted concepts as intended: it shows low dependence between concepts and target outputs and also learns other information apart from concepts to help with its prediction, and thus violates its intended bottlenecking property. As this chapter solely focuses on scalable and interpretable rule learning from propositions, the end-to-end learning from images is not considered, avoiding any potential latent alignment issues. All the experiments in this section take attribute annotations as the ground truth and train a neural DNF-based model to learn classification rules. The end-to-end learning capability of the neural DNF-based models is explored in the later chapters.

The attribute annotation has two components. The 'present' label represents whether the sample has that attribute, where 0 means the attribute is not present and 1 means the attribute is present. The other one is the certainty label which indicates the degree of confidence of the 'present' label being true. The certainty goes from 1 to 4, where 1 is not visible, 2 is guessing, 3 is probable, and 4 is definite. These attribute annotations are generated by non-expert crowd workers and are not consistent throughout the dataset. For example, a red colour might be described by one worker but as rufous by another. This inconsistency results in highly noisy attribute annotations. To de-noise it, the same pre-processing procedure used in the Concept Bottleneck Models paper [88] is applied: it removes the certainty labels and unifies the attribute encoding for different samples in the same

[3] Approaches such as logistic regression, SVMs, and random forests are omitted as they do not yield directly readable rules: logistic regression encodes predictions as a linear scoring function over features with conditional structure; SVMs find a maximal-margin decision boundary, which could be in a high-dimensional kernel space; the ensemble's final prediction of a random forest is governed by majority voting across many trees, which cannot be reduced to a single coherent rule set.



id 2 has_bill_shape::dagger present 0 certainty 3
...
id 92 has_upper_tail_color::white present 1 certainty 3
...
id 312 has_wing_pattern::multi-colored present 0 certainty 4

Figure 5.3: A sample from the CUB-200-2011 dataset before any data pre-processing. The image is ignored and the input to the neural DNF-based models contains the attributes only.

class. The pseudo-code for the pre-processing procedure and further discussion are provided in Appendix B.1.1.

Table 5.1 provides the statistics of the CUB dataset (before and after pre-processing) and the pre-processed subsets used in the experiments. These subsets are used for scalability checks with respect to the number of classes and the number of attributes. They are generated by selecting the first k classes (class 1 to k) in the original dataset, e.g. class 1 to 3 for the CUB-3 subset, and the different number of classes affects the data pre-processing procedure and yields different numbers of attributes in the subsets. For simplicity, from now on, this chapter omits the mention of 'after pre-processing' for each CUB dataset/subset.

Dataset/Subset	No. samples	No. classes	No. attributes
Original CUB-200-2011	11788	200	312
CUB-3	178	3	34
CUB-10	543	10	41
CUB-15	837	15	40
CUB-20	1115	20	48
CUB-25	1402	25	50
CUB-50	2889	50	61
CUB-100	5864	100	82
CUB-200	11788	200	112

Table 5.1: Statistics of different CUB datasets/subsets. Except for 'Original CUB-200-2011', all the other datasets/subsets are pre-processed. The number of samples is the sum of samples from all train, validation and test sets. For dataset names with a suffix '-k', the value of k indicates the number of classes retained in that dataset/subset.

Metrics

Jaccard index score is used to measure both the **correctness** and the **mutual exclusivity** of a neural DNF-based model's output:

$$Jacc(\mathcal{P}, \mathcal{T}) = \frac{|\mathcal{P} \cap \mathcal{T}|}{|\mathcal{P} \cup \mathcal{T}|}$$

where $\mathcal{P}$ is the predicted set of classes and $\mathcal{T}$ is the true set of classes. Given a sample, if the model predicts exactly one class and the correct one, the Jaccard score is 1 as the intersection and the union are identical. If the correct class is not in the prediction, the intersection is empty, leading

to a Jaccard score of 0. If the correct class is predicted but there are more than one class in the prediction, the Jaccard score will be less than 1 since the union is larger than the intersection.

Example 5.2.1: Jaccard index score

Consider a sample with the true class $\mathcal{T} = \{\text{Class 3}\}$, and there are three prediction sets: $\mathcal{P}_1 = \{\text{Class 3}\}$, $\mathcal{P}_2 = \{\text{Class 1, Class 3}\}$, and $\mathcal{P}_3 = \{\text{Class 2, Class 4}\}$. The Jaccard index scores are calculated as follows:

$$Jacc(\mathcal{P}_1, \mathcal{T}) = \frac{|\{\text{Class 3}\}|}{|\{\text{Class 3}\}|} = 1$$
$$Jacc(\mathcal{P}_2, \mathcal{T}) = \frac{|\{\text{Class 3}\}|}{|\{\text{Class 1, Class 3}\}|} = \frac{1}{2}$$
$$Jacc(\mathcal{P}_3, \mathcal{T}) = \frac{|\emptyset|}{|\{\text{Class 2, Class 4}\}|} = 0$$

Experiments

Settings. Two sets of experiments are conducted, with the first set training a plain neural DNF model with binary cross entropy (BCE) (Table 5.2) and cross entropy (CE) (Table 5.3) loss functions on the CUB dataset and subsets, and the second set training a neural DNF-EO model with CE loss function (Table 5.4). The tables show the accuracy, Jaccard index score of the neural DNF-based models after training and at different stages of the post-training pipeline. Specifically, at inference time, the constraint layer (layer C2 in Figure 5.1) is removed from the neural DNF-EO model and the plain neural DNF is kept for classification. The accuracy is computed by taking the arg max of the output logits. The models' bivalent interpretation for each class is determined by a threshold at 0 (True if greater than 0 and False otherwise), and all classes with True interpretations are used as the predicted classes to compute the Jaccard index score.

Dataset	Train Acc.	Train Jacc.	Prune Jacc.	Finetune Jacc.	Threshold Jacc.	Rules Jacc.
CUB-3	1.000	1.000	1.000	1.000	1.000	1.000
CUB-10	0.700	0.417	0.433	0.433	0.433	0.395
CUB-200	0.005	–	–	–	–	–

Table 5.2: Plain neural DNF models trained with binary cross entropy on different dataset/subsets. All metrics are calculated by averaging across the validation set, except for the last column (the extracted rules' Jaccard score), which is computed on the test set. A plain neural DNF cannot learn under BCE with 200 classes, and thus the post-training process is not performed.

Results on plain neural DNF. The first experiment confirms that a plain neural DNF model has trouble learning mutually exclusive rules in multi-class classification. The only successful configuration is using BCE with three classes. Using BCE to train the model effectively treats each rule as independent of the others and tries to push each rule to give discrete predictions (0 and 1). The issue with BCE in multi-class classification is that if there are too many classes, the model can circumvent the loss by predicting all classes as 0 with a small loss, since only one class is true. With only three classes it is relatively easy to learn each rule without sacrificing global performance.

Dataset	Train Acc.	Train Jacc.	Prune Jacc.	Finetune Jacc.	Threshold Jacc.	Rules Jacc.
CUB-3	1.000	0.333	0.368	0.368	0.368	0.318
CUB-10	1.000	0.100	0.117	0.117	0.055	0.057
CUB-200	1.000	0.005	0.003	0.000	–	–

Table 5.3: **Plain neural DNF** trained with **cross entropy** in different dataset/subsets. When using the Jaccard score as the metric during pruning with $\epsilon = 0.005$, all weights are pruned, resulting in no class being predicted.

Dataset	Train Acc.	Train Jacc. (no C2)	Prune Jacc. (no C2)	Finetune Jacc. (no C2)	Threshold Jacc. (no C2)	Rules Jacc.
CUB-3	1.000	1.000	1.000	1.000	1.000	1.000
CUB-10	1.000	1.000	1.000	1.000	1.000	1.000
CUB-15	1.000	1.000	1.000	1.000	1.000	1.000
CUB-20	1.000	1.000	1.000	1.000	1.000	1.000
CUB-25	1.000	1.000	1.000	1.000	0.935	0.933
CUB-50	1.000	1.000	0.982	0.975	0.641	0.636
CUB-100	1.000	1.000	0.967	0.978	0.630	0.632
CUB-200	1.000	1.000	0.967	0.948	0.225	0.225

Table 5.4: **Neural DNF-EO model** trained with **cross-entropy loss** in different CUB subsets/-dataset, at different stages of training and post-training pipeline.

However, Table 5.2 shows that the accuracy drops with the increase in the number of classes. With cross entropy (CE) as the loss function, the model is encouraged to make the correct prediction, but it does not guarantee that only one output is activated. As shown in Table 5.3, the model can predict perfectly by picking the maximum output as the predicted class, achieving perfect accuracy after training, but it suffers heavily on mutual exclusivity as the Jaccard index scores are low across all dataset/subsets. An ideal mutually exclusive output of a disjunctive layer before tanh activation is one where only one element is close to 6 and the others are close to -6. For example, if the output logits are $2, 4, -0.5$ and the target class is 1 (zero-indexed), the softmax-activated output becomes $0.12, 0.87, 0.01$, which results in a low loss but is not completely mutually exclusive, since both $\tanh(2) > 0$ and $\tanh(4) > 0$, leading both classes to be interpreted as True. This experiment shows why using cross entropy alone on plain neural DNF does not guarantee mutual exclusivity, and why the neural DNF-EO model is needed to enforce mutual exclusivity.

Results on neural DNF-EO. The second set of experiments shows that the constraint layer in the neural DNF-EO model can enforce mutual exclusivity when training with CE loss. The plain neural DNF performs correctly with mutual exclusion enforced on all test sets, as shown in the 'Train Jacc. (no C2)' column in Table 5.4. It also provides ASP rules as shown in Listing 5.1 after the post-training pipeline. However, after the CUB-20 subset, the performance gap between a thresholded plain neural DNF ('Threshold Jacc. (no C2)' column) and the original after training ('Train Jacc. (no C2)' column) grows along with the number of classes in the subset. This suggests issues with the thresholding process to maintain the mutual exclusivity in the plain neural DNF (obtained from a trained neural DNF-EO) as the numbers of classes and attributes increase. The issue of the thresholding process will be further discussed in Chapter 7.

```
black_footed_albatross :- not has_bill_colour_black,
    not has_wing_pattern_solid.
laysan_albatross :- has_crown_colour_white.
sooty_albatross :-
    not has_bill_colour_buff, has_crown_colour_black,
    not has_crown_colour_white.
```

Listing 5.1: ASP rules extracted from a trained neural DNF-EO model on the CUB-3 experiment.

Performance comparison between neural DNF-EO and baselines. The performance comparison between neural DNF-EO models and the baselines is separated into two parts: (i) accuracy comparison among neural models (neural DNF-EO models and MLPs) and decision trees (Table 5.5); and (ii) Jaccard index score comparison on the symbolic rules extracted from neural DNF-EO models or directly learned by FastLAS (Table 5.6).

Table 5.5 shows equal accuracy for all models across all experiments, suggesting that the subsets/datasets are not sufficiently challenging to distinguish among the trained plain neural DNF (without constraint layer C2) immediately after training, the MLPs, and decision trees.

The bivalent interpretation of a semi-symbolic node's output takes its tanh output that is greater than 0 as True. For a plain neural DNF, multiple classes can have their tanh activations greater than 0, giving the possibility of multiple classes being True at the same time. And its extracted ASP rules and the rules learned by FastLAS both can give more than one class in an answer set. Thus, the Jaccard index score is used to measure such a requirement of mutual exclusivity. While the plain neural DNF model before the post-training process maintains mutual exclusivity, well-performing rules cannot always be extracted from it, as shown in Table 5.6. On the other hand, the rules learned by FastLAS perform correctly in all settings where they terminate and maintain mutual exclusivity.

Dataset	MLP	Decision Tree (no pruning)	Neural DNF-EO without C2 (after training)
CUB-3	1.000	1.000	1.000
CUB-10	1.000	1.000	1.000
CUB-15	1.000	1.000	1.000
CUB-20	1.000	1.000	1.000
CUB-25	1.000	1.000	1.000
CUB-50	1.000	1.000	1.000
CUB-100	1.000	1.000	1.000
CUB-200	1.000	1.000	1.000

Table 5.5: Accuracy comparison among MLP, decision tree (no pruning) and trained neural DNF-EO model (no constraint layer C2, no post-training processing), across different CUB subsets/dataset.

Scalability comparison. MLPs and decision trees show no scalability issues. However, FastLAS fails to learn from the full dataset of CUB-200, as shown in Table 5.6, where it runs out of memory when computing the opt-sufficient search space. The trained plain neural DNF without any post-training processing learns perfectly in all settings regardless of the number of classes, demonstrating better scalability of neural models when the hypothesis search space is large.

Interpretability compactness comparison. This comparison is only conducted for methods with rule-based interpretability (FastLAS, rules extracted from the neural DNF-EO model and

Dataset	FastLAS	Neural DNF-EO without C2 (after training)	Rules Extracted from Neural DNF-EO without C2
CUB-3	1.000	1.000	1.000
CUB-10	1.000	1.000	1.000
CUB-15	1.000	1.000	1.000
CUB-20	1.000	1.000	1.000
CUB-25	1.000	1.000	0.933
CUB-50	1.000	1.000	0.636
CUB-100	1.000	1.000	0.632
CUB-200	N/A	1.000	0.225

Table 5.6: Jaccard index score comparison across different CUB subsets/dataset, for FastLAS, trained neural DNF-EO model without the constraint layer C2 (no post-training processing), and the rules extracted from the post-training processed neural DNF-EO model without the constraint layer C2. FastLAS runs out of memory for CUB-200.

decision trees). The compactness of the rules is measured by the average and maximum rule length. To measure the rule length of a decision tree, each decision path to a leaf node is treated as a rule, and the number of steps along that path corresponds to the number of body atoms. Figure 5.4 shows the comparison of average and maximum rule length across FastLAS' rules, rules extracted from the neural DNF-EO model (without layer C2), decision trees without pruning, and pruned decision trees with 80% accuracy on train sets. FastLAS learns the most compact rules but has scalability issues with a large hypothesis search space, as it fails to learn with 200 classes present. While the maximum rule length for rules extracted from the plain neural DNF is the highest in most experiments, the average length increases linearly, compared to the decision trees' exponential growth. Rules from the neural DNF-EO model are more compact than the decision tree in half of the settings, especially in experiments with large numbers of classes.

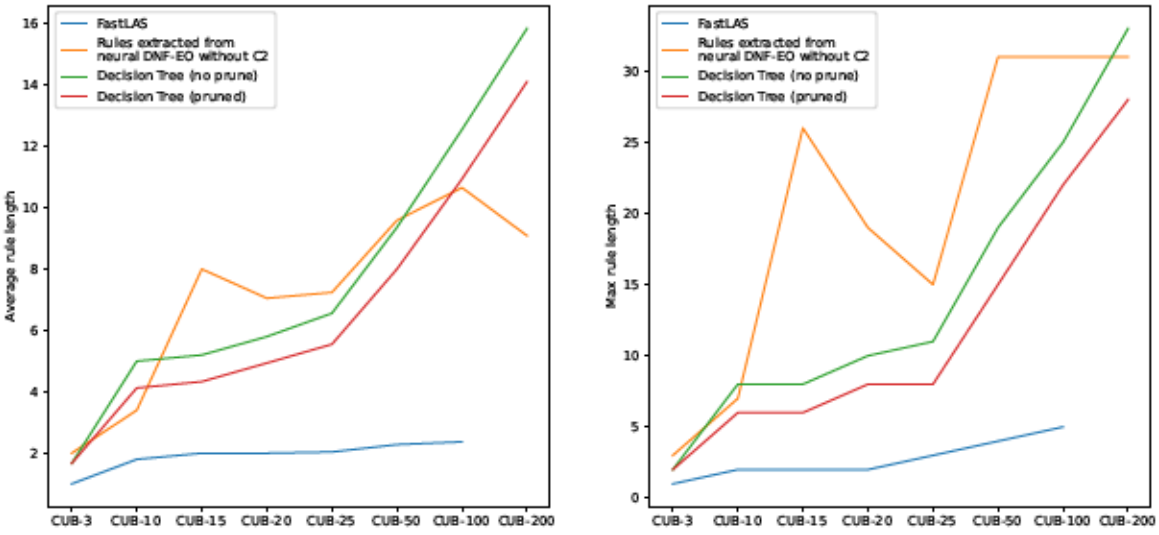


Figure 5.4: Average rule length and maximum rule length comparison across rules learned from FastLAS, extracted rules from trained neural DNF-EO model and decision tree (both pruned and without pruning). The pruned decision trees still maintain an accuracy of 80% in training sets. FastLAS runs out of memory for CUB-200.

5.2.2 Real-world Multi-label Task: TMC Dataset/Subsets

Dataset Description

TMC2007-500 [148] is a text-based multi-label classification dataset based on space shuttle reports, and the goal is to identify different types of anomalies from the text. As a multi-label task, one or more labels can be true at the same time for the same sample, or none of the labels may be true. The dataset provides 500 discrete attributes along with 22 labels, while the reports and actual anomalies are not revealed due to their sensitivity. Listing 5.2 shows a sample from the dataset. The attributes themselves are words, making the input similar to an NLP bag-of-words encoding where each bit of the encoding represents whether a word appears in the report. For this reason, data pre-processing (detailed in Appendix B.2.1) is applied to filter out attributes with low mutual information (MI) with respect to the output label combinations (not on individual labels). Only attributes with MI values greater than or equal to a threshold t are kept. Similar to the CUB dataset, subsets are created for scalability checks and the pre-processing is applied to each subset. Table 5.7 shows the statistics of the datasets/subsets after pre-processing. Different MI thresholds are applied during pre-processing on each subset, resulting in different numbers of attributes used in each subset.

```
{
    "sample_id": "18643",
    "present_attributes": [2, 92, ..., 480, 481],
    "present_labels": [1, 5, 7, 12]
    "present_attributes_text_mapping": {
        "2": "above",
        "92": "climb",
        ...,
        "480": "we",
        "481": "weather"
    }
}
```

Listing 5.2: A sample from the TMC2007-500 dataset before any data pre-processing.

Dataset/Subset	No. samples	No. labels	No. attributes	MI t
Original TMC2007-500	28596	22	500	N/A
TMC-3	16947	3	59	0.01
TMC-5	18565	5	60	0.02
TMC-10	21454	10	34	0.04
TMC-15	24318	15	80	0.04
TMC-22	28538	22	103	0.05

Table 5.7: Statistics of different TMC datasets/subsets. Except for 'Original TMC2007-500', all the other datasets/subsets are pre-processed. The number of samples is the sum of samples from all train, validation and test sets. For dataset names with a suffix '-k', the value of k indicates the number of labels retained in that dataset/subset.

Experiments

Settings. Macro F1 is used as the metric for the performance. Neural DNF models are trained with binary cross-entropy loss, and the baselines are 2-layer MLPs trained with binary cross-entropy loss, decision trees without limiting depth and FastLAS.

Results on neural DNF-EO. Table 5.8 shows the performance of neural DNF models and the extracted rules across different TMC subsets/dataset. The neural DNF model struggles to learn as the number of attributes and labels increases in the subsets/dataset. However, this may be attributable to the lack of general patterns or regularities in the TMC dataset/subsets. This issue is further discussed later when comparing to other baselines. On the other hand, the rules extracted through the post-training process show very similar performance to the corresponding trained neural DNF model. Compared to the CUB experiments, the thresholding process does not need to maintain mutual exclusivity across different outputs and does not cause a significant performance drop when discretising the continuous weights.

Comparison between neural DNF models and baselines. There is an increasing performance gap between the neural DNF models and MLPs as the numbers of attributes and labels increase. Figure 5.9 shows the change in macro F1 and the δ magnitude in the neural DNF model's symbolic layers over 300 epochs of training on TMC-22. The neural DNF model achieves a macro

Dataset	Train macro F1	Prune macro F1	Finetune macro F1	Threshold macro F1	Rules macro F1	MLP + BCE Macro F1	Decision Tree (no pruning) Macro F1
TMC-3	0.745	0.764	0.739	0.754	0.754	0.817	0.925
TMC-5	0.615	0.649	0.648	0.635	0.635	0.746	0.894
TMC-10	0.391	0.380	0.370	0.362	0.362	0.539	0.825
TMC-15	0.177	0.143	0.100	0.097	0.097	0.538	0.863
TMC-22	0.094	0.086	0.093	0.094	0.094	0.545	0.865

Table 5.8: Macro F1 of neural DNF models after training with BCE and at different stages of the post-training pipeline, MLPs trained with BCE, and decision trees without limiting depth, in different TMC subsets/dataset.

F1 lower but still in a similar range of values to the MLP's (around 0.45 compared to MLP's 0.545) when the logical biases are not heavily enforced. But as the δ magnitude increases, especially over 0.5, the performance of the model drops and never recovers. This suggests that the logical biases in the semi-symbolic layers make it more difficult for the neural DNF model to learn. Chapter 7 introduces a solution to mitigate the problem of δ magnitude and training difficulty.

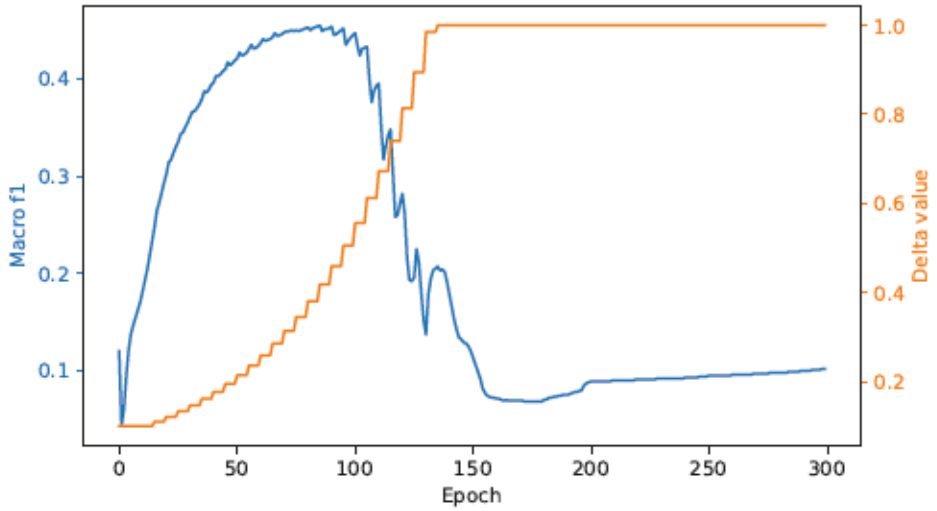


Table 5.9: Macro F1 and the δ magnitude of a neural DNF model in TMC-22 dataset over 300 epochs of training. The drop in macro F1 occurs around when the δ magnitude is around 0.5.

The decision tree significantly outperforms both neural DNF models and MLPs across all subsets/datasets, as shown in Table 5.8. This superior performance, especially as the numbers of attributes and labels increase, suggests that the dataset may lack general patterns and requires an architecture with more parameters; however, both the MLP and the neural DNF models risk overfitting when the number of parameters becomes too large. An MLP with more and wider hidden layers can approximate increasingly complex functions. Similarly, for the neural DNF-based model, an infinite number of conjunctions (i.e., infinite width of the conjunctive layer) would theoretically allow it to encode every possible rule for each label. However, this would lead to learning highly specific rules that activate for only small subsets of data points. By constraining the number of conjunctions or hidden layer width, these neural models are forced to generalise across larger sample groups. When datasets lack general patterns that can be captured with low-dimensional features, models require

wider hidden layers (and maybe more layers as well) to expand the feature space and inevitably increase the overfitting risk. On the other hand, decision trees split the input space recursively, and without depth constraints, they can encode every training case through nested conditionals and overfit the data. Table 5.10 compares the trainable parameters in MLP/neural DNF models against the number of non-leaf nodes (split nodes) in unconstrained decision trees. Although the number of non-leaf nodes is not directly comparable to the number of learnable parameters in MLP/neural DNF models, it can be treated as a proxy for the number of parameters since it encodes a split condition that directs the decision path. The unlimited-depth decision tree has more non-leaf nodes than the number of learnable parameters in its corresponding MLP/neural DNF counterparts in all experiments, showing fine-grained decision making to perform well in the subsets/dataset. When the depth is limited, the decision tree suffers from performance degradation as shown in Figure 5.5. However, even when the depth is limited, the decision tree still outperforms the MLP/neural DNF models most of the time (except when the depth is limited to 10). This suggests that the unconstrained decision tree's superior performance might stem from both its greater model capacity and also some degree of overfitting, and that the dataset's inherent complexity requires more parameters for effective learning in neural models.

The last baseline approach, FastLAS, fails in *every* TMC subset/dataset, suffering from the same memory issue as the CUB-200 experiment. In a multi-class setting, the mutual exclusivity constraint can help reduce the hypothesis search space, but without the constraint in the multi-label setting, the hypothesis search space becomes exponentially larger since every label can be fired at the same time, causing a memory issue for FastLAS.

Dataset	MLP/Neural DNF Architecture (Input, Hidden, Output)	Num. Parameters (Excluding bias)	Decision Tree Max. depth	No. Non-leaf Nodes
TMC-3	(59, 9, 3)	558	54	1461
TMC-5	(60, 15, 5)	975	50	3436
TMC-10	(34, 30, 10)	1320	34	7715
TMC-15	(80, 34, 15)	4275	67	9679
TMC-22	(103, 66, 22)	8250	66	11878

Table 5.10: MLP and neural DNF model's trainable parameters count, comparing to the number of non-leaf nodes in a decision tree without limiting depth, in different TMC subsets/dataset. A non-leaf node of a decision tree encodes a split condition that directs the decision path.

5.2.3 Synthetic Dataset

The last set of experiments runs on synthetic multi-class and multi-label datasets to verify the capabilities of the neural DNF-based models.

Dataset Generation. For each synthetic dataset, a set of rules is randomly generated as its ground truth. With M attributes and N classes/labels, a set of C conjunctions that can be used by the classes/labels is randomly generated, with each conjunction's rule bodies containing a random subset of the attributes (either positive or negative literals). Each class/label is associated with only one conjunction. This choice is to simplify the generation process for a multi-class setting due to the mutual exclusivity constraint: by associating each class to only one conjunction, as long as the

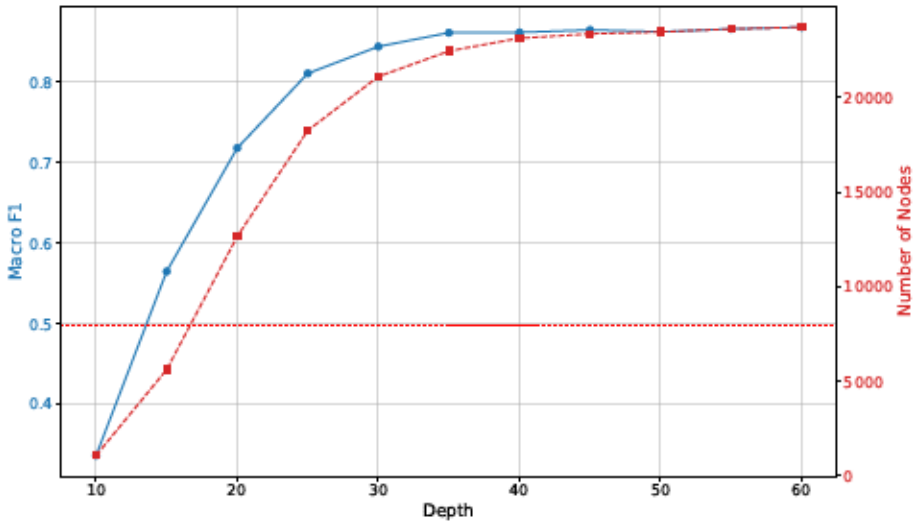


Figure 5.5: Evaluation of the decision tree's performance on TMC-22 dataset with different maximum depths. The red dotted line represents 8000 number of non-leaf nodes that is similar to the number of parameters in an MLP/neural DNF model. Both the macro F1 and the number of nodes decrease as the depth decreases. When the decision tree's maximum depth is limited to 15, it has 2,810 non-leaf nodes and achieves a test macro F1 of 0.565, still better than the MLP's performance but close to it.

conjunctions are unique and with no intersection between the rule bodies, the mutual exclusivity constraint is guaranteed to be satisfied. After the ground truth rules are generated, samples are generated by rejection sampling: for each sample of (X, y) pair, the input X is randomly generated as either True or False for each attribute and the output y is the classes/labels in the answer set computed by clingo [126] using the ground truth rules and the input X. In a multi-label setting, a sample will be automatically accepted since there is no constraint on its outputs. Note that no noise is added to the samples, that is, the samples' outputs are never altered and also match the answer sets computed by clingo. The samples are split into training, validation and test sets in $64:16:20$ ratio. Table 5.11 shows the statistics of the synthetic datasets.

Dataset Name	Type	No. Attributes (M)	No. Classes/Labels (N)	No. Conjunctions in the Pool (C)	Total No. Samples
MC-3	Multi-class	15	3	9	10000
MC-25	Multi-class	150	25	75	10000
ML-3	Multi-label	27	3	9	10000
ML-25	Multi-label	150	25	75	10000

Table 5.11: Statistics of the synthetic datasets.

Settings. Following the same training and post-training process, but without the finetuning stage, neural DNF-based models are trained and evaluated on each of the synthetic datasets. Neural DNF-EO models are trained with cross-entropy loss for the multi-class setting and vanilla neural DNF models are trained with binary cross-entropy loss for the multi-label setting. Since the purpose is to verify the capabilities of the neural DNF-based models, no baseline models are used for comparison.

Results. The experiments on the synthetic datasets confirm the neural DNF-based models' capabilities of learning rules from datasets with underlying patterns, as shown in Table 5.12 and Table 5.13. The neural DNF-based models learn well after training in all experiments, while the problem of multi-class classification performance dropping after thresholding persists, matching the same issue in the real-world dataset experiments in Section 5.2.1.

Synthetic Dataset	After train	After prune	After threshold	Extracted Rules
MC-3	1.000	1.000	0.981	0.974
MC-25	0.995	0.991	0.227	0.226

Table 5.12: Neural DNF-EO models' Jaccard index score on different synthetic multi-class datasets at different stages of the pipeline.

Synthetic Dataset	After train	After prune	After threshold	Extracted Rules
ML-3	1.000	0.995	0.995	0.995
ML-25	1.000	0.967	0.987	0.987

Table 5.13: Neural DNF models' macro F1 score on different synthetic multi-label datasets at different stages of the pipeline.

5.3 Discussion

This section analyses the neural DNF-based models in the three areas of *classification performance*, *interpretability*, and *scalability* to compare against the baselines of MLPs, decision trees and FastLAS [91]. To summarise, (i) the neural DNF-based model can perform at a comparable level to an MLP and a decision tree after training; (ii) it provides interpretability as a neural model by allowing the extraction of ASP rules, and the extracted rules are on average more compact than a decision tree's decision paths; (iii) it also scales better than FastLAS when the hypothesis search space is large.

5.3.1 Classification Performance

When there exist underlying general patterns in the dataset that can be learned without extreme fine-grained decision making, such as the CUB dataset/subsets in Section 5.2.1 and the synthetic datasets in Section 5.2.3, the neural DNF-based models without any post-training processing perform comparably to MLPs and decision trees. In particular, the EO constraint layer successfully enforces mutual exclusivity on the trained plain neural DNF model's output in the multi-class classification tasks, achieving the design goal of the neural DNF-EO model. In the TMC experiments, both the neural DNF models and MLPs cannot generalise well, while decision trees without a depth limit overfit with numerous decision nodes. The TMC experiments also show a larger difference in performance between the neural DNF models and MLPs, and the training loss plot of the neural DNF model in Figure 5.9 suggests that the logical biases in the semi-symbolic layers of the neural DNF model increase the difficulty of training.

5.3.2 Interpretability

The logically constrained bias in the semi-symbolic layers makes it possible to extract symbolic rules from trained neural DNF-based models through post-training processes, contrasting with the lack of interpretability in MLPs. However, as shown in the CUB experiments (Table 5.4), the current process does not always result in a successful rule extraction for multi-class classification. The thresholding process is the current bottleneck of the pipeline, as it cannot maintain mutual exclusivity across a large number of classes. The neural DNF-EO model may also rest on an 'unstable equilibrium' (Figure 5.6) after training: an arbitrarily small change made to the model during the post-training process could push it towards a different sub-optimal or poor solution, demonstrating the model's brittleness with respect to interpretability. Similar thresholding issues with the neural DNF-EO model reappear in Section 7.5, which may be related to this instability hypothesis. Verification of the unstable equilibrium conjecture remains as future work.

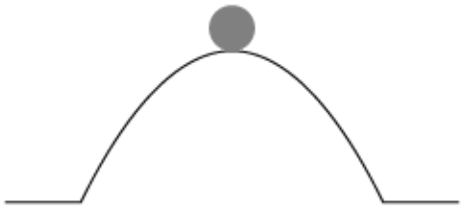


Figure 5.6: An example of an unstable equilibrium.

The extracted rules are also not guaranteed to be in the most compact form compared to FastLAS (Figure 5.4). FastLAS with a scoring function on body length will guarantee the most compact hypothesis, while the neural DNF-based models do not have a strong signal to learn the most compact rules, which may further restrict gradient-based learning. Although the auxiliary loss $|w||6 - |w||$ is included during training and the model is pruned at post-training processes, it does not necessarily reject hypotheses with longer rules. On the other hand, decision trees show less compactness than rules extracted from the neural DNF-based models, and also are prone to overfitting.

5.3.3 Scalability

Theoretically, the neural DNF-based models scale better than FastLAS, and this is borne out by the experiments. The conjunctive and disjunctive layers of the neural DNF-based model create a continuous hypothesis space that covers rules with any combination of body atoms, and through gradient descent, the model moves towards an approximated hypothesis. FastLAS first needs to compute a hypothesis sub-space that guarantees to contain the optimal solution and may run out of memory during this computation process, which happens in the CUB-200 experiment and all TMC experiments. Comparing the neural DNF-based model to MLP and decision tree, all three can be trained without memory issues and no background knowledge is needed, regardless of the input-output size. However, training the neural DNF-based model is more difficult than training an MLP (Figure 5.9). The neural DNF-based model has fewer trainable parameters than an MLP under the same number of neurons, and the logical bias in the semi-symbolic layers imposes strong constraints on the model. Apart from the learning difficulty, the thresholding process in the post-training pipeline does not yield perfect rules from trained models with more than 20 classes (Table 5.4).

5.4 Conclusion

This chapter introduces the neural DNF-EO model that enforces mutual exclusivity on the model's outputs using a frozen constraint layer in multi-class classification tasks. The multi-class experiments demonstrated the insufficiency of using a cross-entropy loss to train a plain neural DNF model for mutually exclusive outputs, and in contrast demonstrated the capabilities of the neural DNF-EO model as an improvement. Despite its shortcomings in multi-class classification tasks, the neural DNF model is still suitable for independent rule learning in multi-label classification tasks. Both the neural DNF-EO and the neural DNF model are shown to perform at a comparable level to the MLP in real-world classification tasks. Furthermore, the scalability of the neural DNF-based models is shown to be better than the state-of-the-art symbolic rule learner FastLAS over large search spaces. Compared to decision trees, the neural DNF-based models learn more compact rules and provide end-to-end training capabilities with other differentiable models.

Chapter 6

Neural DNF-MT Model for Reinforcement Learning

In addition to classification, another highly relevant domain for neuro-symbolic learning is reinforcement learning, where the demand for model interpretability is driven by safety and ethical considerations. Remarkable progress has been made in deep reinforcement learning (RL) with the advancement of neural networks. Since the demonstration of impressive performance in complex games like Go [145] and Dota 2 [19], significant effort has been made to utilise deep RL approaches for solving real-life problems, such as segmenting surgical gestures [59] and providing treatment decisions [163]. Reinforcement Learning from Human Feedback (RLHF) [14, 118] has emerged as a crucial technique for aligning LLMs with human preferences, where models are first fine-tuned using supervised learning on human demonstrations, then further optimised using RL algorithms to maximise rewards based on human preference rankings. However, safety and ethical considerations necessitate interpretable models. In the EU's AI Act, systems used in areas such as healthcare fall into the high-risk category, requiring both a high level of accuracy and a method to explain and interpret their output [1]. Therefore, the 'black-box' nature of neural models becomes a concern when using them for such high-stakes decisions in healthcare [70].

Various neuro-symbolic approaches address the lack of interpretability in deep RL. The term 'symbolic' in this thesis refers to methods that offer *logical rule representations*, in contrast to program synthesis approaches [155, 154, 24] that offer *programmatic representations* with forms of logic. Some of these neuro-symbolic methods [77, 39] rely on manually engineered inductive bias to restrict the search space and thus limit the rules they can learn. Others [84, 170] without predefined inductive bias associate weights with predicates but require pre-trained components to parse observations to predicates [84] or a special critic for training [170].

This chapter proposes a variant of the neural DNF model, called the neural DNF-MT model, for learning interpretable and editable policies. The neural DNF-MT model still utilises semi-symbolic nodes but with modifications that support probabilistic representation for policy learning. The model is completely differentiable and supports integration with deep actor-critic algorithms. It can also be used to distil policies from other neural models. From trained neural DNF-MT actors, bivalent logic programs for deterministic policies or probabilistic logic programs for stochastic policies can be extracted. These interpretable logical representations are close approximations of the learned

models. Furthermore, owing to the nature of the bidirectional neural-bivalent-logic translation, manual policy intervention on the model is possible. By modifying the bivalent logical program and porting it back to the neural model, the tensor operations and environment parallelism for fast inference are leveraged. Compared to existing works, the neural DNF-MT model does not rely on rule templates or mode declarations. Furthermore, the model is trained with a simple MLP critic and supports trainable preceding layers to extract relevant features from complex observations, such as multi-dimensional matrices.

The rest of this chapter includes material from our paper [16].

6.1 Issues of Existing Neural DNF-based Models

Unlike multi-class classification, where each sample has a single deterministic class, an RL actor seeks to approximate the optimal policy with potentially arbitrary action probabilities [150]. It is possible for a domain to have an optimal deterministic policy and for the RL algorithm to approach it with an 'almost deterministic' policy, where for each state the optimal action's probability is significantly greater than the others (i.e. a single almost-1 value vs all the rest close to 0). In this case, the actor almost always chooses a single action, similar to a multi-class classification model predicting a single class. A trained neural DNF-based model representing such a policy should be interpreted as a bivalent logic program representing the nearest deterministic policy. When preserving the probabilities encoded within the trained neural DNF-based actor without approximating it with the nearest deterministic policy, its interpretation should be captured as a probabilistic logic program that expresses the action distributions. There is no way to achieve both of these objectives with the neural DNF and neural DNF-EO models, since their interpretation frameworks do not satisfy two forms of mutual exclusivity: (a) *probabilistic mutual exclusivity* when interpreted as a stochastic policy, and (b) *logical mutual exclusivity* when interpreted as a deterministic policy. This section first formalises the logic system represented by neural DNF-based models (Definition 4.1.1) and then defines the two mutual exclusivities possible in this logic system (Definition 6.1.1 and 6.1.2).

Definition 6.1.1 (Logical mutual exclusivity). *Given the final activation of a neural DNF-based model for N classes $\mathbf{y} \in (-1,1)^N$ and its bivalent logic interpretation $\mathbf{b} \in \{\bot,\top\}^N$ as in Definition 4.1.1, the model satisfies* ***logical mutual exclusivity*** *if there is exactly one b_i that is $\top$:*

$$\models \left(\bigvee_{i \in \{1..N\}} b_i \right) \wedge \left(\bigwedge_{i,j \in \{1..N\}, i<j} \neg(b_i \wedge b_j) \right)$$

Definition 6.1.2 (Probabilistic mutual exclusivity). *A probabilistic interpretation of GBL is a function $f_p : (-1,1) \to (0,1)$ that maps each belief y_i to probability p_i that b_i holds as true. Formally,*

$$p_i = f_p(y_i) = \Pr(b_i = \top | y_i)$$

A neural DNF-based model satisfies ***probabilistic mutual exclusivity*** *if the interpreted probabilities $\mathbf{p} = (p_1, \ldots, p_N)^T \in (0,1)^N$ associated with its activations $\mathbf{y} \in (-1,1)^N$ under probabilistic*

interpretation f_p sum to 1. That is:

$$\sum_{i}^{N} f_p(y_i) = 1$$

To be used for interpretable policy learning, a neural DNF-based model must guarantee the following properties:

P1: The model provides a probabilistic mutually exclusive interpretation (Definition 6.1.2) and can be interpreted as a probabilistic logic program (such as ProbLog [37], see Listing 6.2 as an example),

P2: When the *optimal policy is sufficiently close to deterministic*, the model can also be interpreted as a bivalent logic program (such as ASP [95], see Listing 6.1 as an example) that satisfies logical mutual exclusivity (Definition 6.1.1).

A trained neural DNF model from [28] does not provide probabilistic interpretation or guarantee logical mutual exclusivity in its bivalent interpretation, and thus fails P1 and P2. A trained neural DNF-EO model from the previous chapter satisfies P2 via its constraint layer but fails to provide probabilistic interpretation for P1. To address these requirements, this chapter proposes a new model called neural DNF-MT and post-training processing steps that translate a trained neural DNF-MT model into a ProbLog program and, where applicable, into an ASP program. The proposed model satisfies both properties above. The following section describes how the proposed model is constructed and trained, and how logical interpretations that satisfy the properties above are extracted from the model.

6.2 Methodology

6.2.1 Mutex-tanh Activation and Neural DNF-MT

Let $\mathbf{d} \in \mathbb{R}^N$ be the output vector of a disjunctive semi-symbolic layer before any activation function and $d_k \in \mathbb{R}$ be the output of the k^{th} disjunctive node. Using the softmax function, the new activation function mutex-tanh is defined as:

$$\begin{aligned} \text{softmax}(\mathbf{d})_k &= \frac{e^{d_k}}{\sum_{i}^{N} e^{d_i}} \\ \text{mutex-tanh}(\mathbf{d})_k &= 2 \cdot \text{softmax}(\mathbf{d})_k - 1 \end{aligned} \tag{6.1}$$

With the mutex-tanh activation function, the neural DNF-MT model is constructed with a semi-symbolic conjunctive layer with a tanh activation function and a disjunctive semi-symbolic layer with the mutex-tanh activation function:

$$\begin{aligned} \mathbf{c} &= \tanh(\mathbf{W_c x} + \beta_{\mathbf{c}}) && \text{Output of conj. layer} \\ \mathbf{d} &= \mathbf{W_d c} + \beta_{\mathbf{d}} && \text{Raw output of disj. layer} \\ \tilde{\mathbf{y}} &= \text{mutex-tanh}(\mathbf{d}) && \text{mutex-tanh output of disj. layer} \end{aligned}$$

where $\mathbf{W_c}$ and $\mathbf{W_d}$ are trainable weights, and $\beta_{\mathbf{c}}$ and $\beta_{\mathbf{d}}$ are the logical biases calculated as Equation (4.2). Note that $\tilde{\mathbf{y}} \in (-1, 1)^N$ shares the same codomain as the disjunctive layer's tanh output

$\hat{\mathbf{y}} = \tanh(\mathbf{d})$. The disjunctive layer's bivalent interpretation $\hat{\mathbf{b}}$ still follows the GBL interpretation in Definition 4.1.1 and uses $\hat{\mathbf{y}}$, with $\hat{b_i} = \top$ when $\hat{y_i} > 0$ and $\bot$ otherwise.

To satisfy P1, the probability $\tilde{\mathbf{p}}$ is computed as:

$$\tilde{\mathbf{p}} = (f_p(\tilde{y}_1), \ldots, f_p(\tilde{y}_N))^T \quad \text{where} \quad f_p(\tilde{y}_i) = \frac{\tilde{y}_i + 1}{2} \tag{6.2}$$

By construction, $\tilde{\mathbf{p}} \in (0,1)^N$, and $\sum_k^N \tilde{p_k} = 1$ from Equation (6.1) satisfies probabilistic mutual exclusivity.

6.2.2 Architecture and Training of Neural DNF-MT as Actor with PPO

Using the PPO algorithm [138], a neural DNF-MT actor is trained with an MLP critic. The input to the neural DNF-MT actor must be in $[-1,1]^I$. Any discrete observation is converted into a bivalent vector representation, as shown in Figure 6.1a. If the observation is complex, as shown in the experiment in Section 6.3.4, an encoder can be added before the neural DNF-MT actor to invent predicates in GBL form. In this case, the encoder output acts as input to the neural DNF-MT actor and the MLP critic, as shown in Figure 6.1b.

This section presents the overall training loss of the actor-critic PPO with a neural DNF-MT actor, which consists of multiple loss terms. The base training loss component matches that from PPO [138]:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) + \kappa_1 L^{\text{value}}(\theta) - \kappa_2 S[\pi_\theta](s_t) \right] \tag{6.3}$$

where $\kappa_1, \kappa_2 \in \mathbb{R}$ are hyperparameters, $L^{\text{CLIP}}(\theta)$ is the clipped surrogate objective, $S[\pi_\theta](s_t)$ is the entropy of the actor in training, and $L^{\text{value}}(\theta)$ is the value loss. The detailed explanations for each term are provided in the background section on PPO in Section 2.4.2. The action probability output of the neural DNF-MT actor defined in Equation (6.2) is used to calculate the probability ratio in L^{CLIP} and the entropy term $S[\pi_\theta](s_t)$.

The following auxiliary losses are added to facilitate the interpretation of the neural DNF-MT model into rules:

$$L^{(1)}(\theta) = \frac{1}{N_F} \sum_i^{N_F} |1 - |f_i|| \tag{6.4}$$

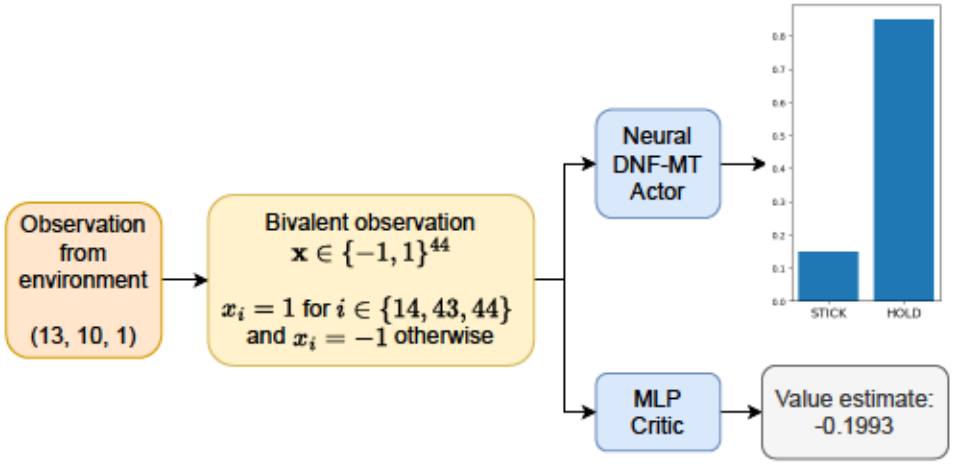
$$L^{(2)}(\theta) = \frac{1}{|\theta_{\text{disj}}|} \sum |\theta_{\text{disj}} \cdot (6 - |\theta_{\text{disj}}|)| \tag{6.5}$$

$$L^{(3)}(\theta) = \frac{1}{N_C} \sum_i^{N_C} |1 - |c_i|| \tag{6.6}$$

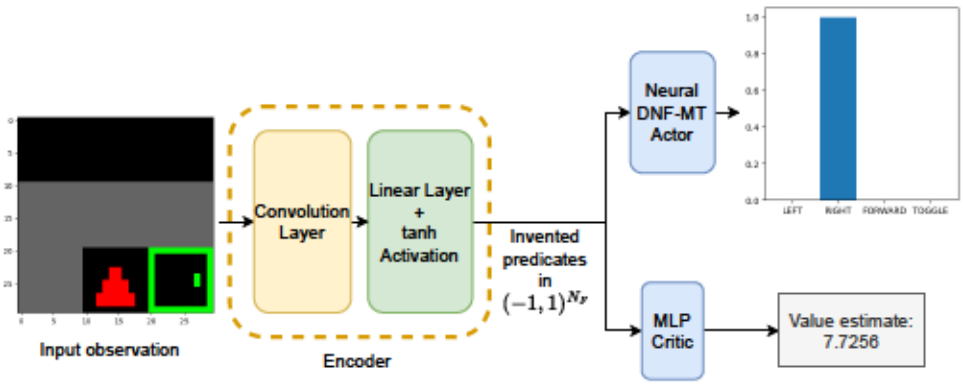
$$L^{(4)}(\theta) = -\sum_i^{N} \left[\hat{p}_i \log\left(\frac{\hat{y}_i + 1}{2}\right) + (1 - \hat{p}_i) \log\left(1 - \frac{\hat{y}_i + 1}{2}\right) \right] \tag{6.7}$$

where f_i is the invented predicate and N_F is the number of outputs of an encoder when there is an encoder. c_i is the tanh activation of the i^{th} conjunctive node and N_C is the number of conjunctive nodes. $\hat{p}_i$ is the predicted action probability of the i^{th} disjunctive node and $\hat{y}_i$ is the tanh output of the i^{th} disjunctive node.

Equation (6.4) is used when there is an encoder before the neural DNF-MT actor for predicate



(a) Neural DNF-MT model as an actor in actor-critic PPO, in an example environment with discrete observations.



(b) Neural DNF-MT model as an actor in actor-critic PPO, in an example environment with complex observations, such as an image-like multi-dimensional matrix.

invention. It enforces the predicates' activations to be close to ± 1 so that they are stronger beliefs of true/false. Equation (6.5) is a weight regulariser to encourage the disjunctive weights to be close to ± 6 (the choice of ± 6 is to saturate tanh, as $\tanh(\pm 6) \approx \pm 1$). Equation (6.6) encourages the tanh output of the conjunctive layer to be close to ± 1. Equation (6.7) is the key term to satisfy P2, pushing for bivalent logical interpretations for deterministic policies. This term mimics a cross-entropy loss between each mutex-tanh output and corresponding individual tanh outputs of the disjunctive layer, pushing the probability interpretations of the tanh outputs (i.e. $(\hat{y}_i + 1)/2$) towards their action probability $\tilde{p}_i$ counterparts. If the optimal policy is deterministic, all $\tilde{p}_i$ will be approximately 0 except for one, which is close to 1. Each $\hat{y}_i$ is pushed towards ± 1, and only one will be close to 1, thus having exactly one bivalent interpretation $b_i = \top$ and satisfying logical mutual exclusivity.

Finally, the overall training loss is defined as:

$$L(\theta) = L^{\text{PPO}}(\theta) + \sum_{i \in \{1,2,3,4\}} \lambda_i L^{(i)}(\theta) \tag{6.8}$$

where $\lambda_i \in \mathbb{R}, i \in \{1,2,3,4\}$ are hyperparameters. In particular, λ_1 is set to 0 when there is no encoder needed for predicate invention.

Apart from the training objective, the training technique of δ delayed exponential scheduling is also adopted (Equation (2.7) in Section 2.3.1) for neural DNF-MT models, such that the logical bias is gradually increased throughout training for smoother learning.

6.2.3 Post-training Processing

The post-training process extracts either a ProbLog program for a stochastic policy or an ASP program for a close-to-deterministic policy from a trained neural DNF-MT actor, where the logic program is a close approximation of the model. Following a similar structure to the post-training process in Section 2.3.3, it consists of multiple stages that include pruning and translation. However, some steps are modified to accommodate policy learning. The full post-training process for interpreting a trained neural DNF-MT actor is shown in Figure 6.2.

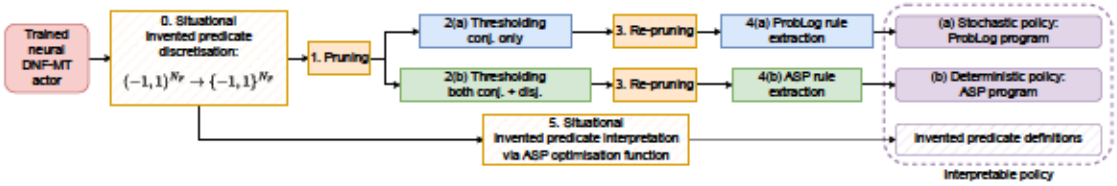


Figure 6.2: Post-training processing to extract an interpretable logical policy from a trained neural DNF-MT actor. There are two branches: one with sub-label (a) for extracting a stochastic policy in ProbLog and the other with sub-label (b) for extracting a deterministic policy in ASP.

(1) Pruning: This step repeatedly passes over each edge that connects an input to a conjunction or a conjunction to a disjunction, and removes any edge that can be removed (i) without changing the learned trajectory (for close-to-deterministic policies) or (ii) without shifting any action probability for any state more than some threshold τ_{prune} from the original learned policy (for stochastic policies). Any unconnected nodes are also removed. Because the removal of an edge in a later pass may render previously retained edges redundant, the procedure iterates over the network multiple times rather than terminating after a single sweep. It concludes once a full pass yields no further removals, indicating that the model has reached a stable state in which no remaining edge can be deemed redundant. In practice, conjunctive-node-to-disjunction-node edges are processed first before the input-to-conjunctive-node edges, and $\tau_{\text{prune}} = 1e-3$ is set for the removal check. The pruning process is formalised in Algorithm 9.

(2) Thresholding: This is the discretisation process mentioned in Section 4.1.2 that converts a semi-symbolic layer's weights from $\mathbb{R}$ to values in $\{-6,0,6\}$ and enables the *neural to bivalent logic translation* described in Section 4.2. Given some threshold $\tau \in \mathbb{R}_{\geq 0}$, a new weight is computed as $w'_{\text{k}ij} = 6 \cdot \mathbb{1}(|w_{\text{k}ij}| \geq \tau) \cdot \text{sign}(w_{\text{k}ij})$, where $\text{k} \in \{\text{c}, \text{d}\}$ indicates the type of the node (conjunctive or disjunctive). The selection of τ should maintain the model's trajectory/action probability, subject to the same checks used in pruning. For a thresholded node with at least one non-zero weight, its tanh activation is replaced with step function $h(x) = 2 \cdot \mathbb{1}(x > 0) - 1$, changing its output's range to $\{-1,1\}$. The thresholding process is applied differently to the disjunctive layer depending on the nature of the policy desired. This step is formalised in Algorithm 10.

(2.a) **For stochastic policies**: Only the conjunctive layer is thresholded, i.e. choosing a value of τ, updating only its weights and changing the activation function. The disjunctive layer still outputs action probabilities.

(2.b) **For deterministic policies**: Thresholding is applied to both the conjunctive and disjunctive layers: a single value τ is chosen and applied in both layers' weight update, and both layers have their tanh activation replaced with the step function. This process is only possible if the model satisfies P2.

(3) Re-pruning: The pruning process from Step 1 is repeated.

(4) Logical rules extraction: All nodes (conjunctive and disjunctive) are converted into some form of logical rules. The thresholding process guarantees that all conjunctive nodes can be translated into bivalent logic representations. For a conjunctive node c_j, the set $\mathcal{X}_j = \{i \in \{1..I\} | w'_{cij} \neq 0\}$ is considered, and $|\mathcal{X}_j| \neq 0$. $\mathcal{X}_j$ is partitioned into subsets $\mathcal{X}_j^+ = \{i \in \mathcal{X}_j | w'_{cij} = 6\}$ and $\mathcal{X}_j^- = \{i \in \mathcal{X}_j | w'_{cij} = -6\}$, and translate c_j to an ASP rule of the form $conj_j \leftarrow \bigwedge_{i \in \mathcal{X}_j^+} atom_i, \bigwedge_{i \in \mathcal{X}_j^-} (\text{not } atom_i)$, where $atom_i$ is an atom for input x_i. The disjunctive nodes are interpreted differently depending on the desired policy type.

(4.a) **Stochastic policy - ProbLog rules**: To represent mutually exclusive action probabilities, ProbLog's annotated disjunction representation is chosen to specify the set of action probabilities for each action. An annotated disjunction of the form $p_1 :: f_1; \ldots; p_n :: f_n \leftarrow body$ associates a probability p_n to each individual ground probabilistic fact f_n if $body$ is true, and $\sum_{i=1}^{n} p_i = 1$, and only one of the f_ns is true. Annotated disjunctions match the requirement for probabilistic mutual exclusivity of the neural DNF-MT model and also the actor's behaviour of choosing exactly one action when interacting with the environment. To translate the disjunctive layer with the mutex-tanh activation, each unique achievable activation of the conjunctive layer $\mathbf{c}^{(m)} \in \{-1, 1\}^{C'}$ with $1 \leq m \leq 2^{C'}$, where C' is the number of remaining conjunctive nodes after pruning and may differ from the initial choice of C, forms the body of a unique annotated disjunction of the form $p_1 :: action_1; ...; p_N :: action_N \leftarrow \bigwedge_{i \in \mathcal{C}^{(m)+}} conj_i, \bigwedge_{i \in \mathcal{C}^{(m)-}} (\backslash + conj_i)$, where $\mathcal{C}^{(m)+} = \{i | c_i^{(m)} = 1\}$, $\mathcal{C}^{(m)-} = \{i | c_i^{(m)} = -1\}$, and $p_j = (\tilde{y}_j^{(m)} + 1)/2$ (the probability assigned to the j^{th} action in the disjunctive activation for the m^{th} unique activation). Such annotated disjunctions are computed for all unique conjunctive activations. A pseudocode for this step is shown in Algorithm 11, and examples of ProbLog rules extracted from the trained neural DNF-MT actor are shown in the later sections.

(4.b) **Deterministic policy - ASP rules**: Since the disjunctive layer is also thresholded, each disjunctive node is translated into a normal clause. For a disjunctive node d_j, the set $\mathcal{C}_j = \{i \in \{1..C'\} | w'_{dij} \neq 0\}$ is considered, and $|\mathcal{C}_j| \neq 0$. $\mathcal{C}_j$ is partitioned into subsets $\mathcal{C}_j^+ = \{i \in \mathcal{C}_j | w'_{dij} = 6\}$ and $\mathcal{C}_j^- = \{i \in \mathcal{C}_j | w'_{dij} = -6\}$, and translate d_j to a formula of the form $disj_j \leftarrow (\bigvee_{i \in \mathcal{C}_j^+} conj_i) \vee (\bigvee_{i \in \mathcal{C}_j^-} (\text{not } conj_i))$. In practice, the formula is represented as multiple rules with the same head in ASP. Listing 6.1 shows an example of ASP rules, and Algorithm 12 shows the pseudocode for this step.

If there is an encoder before the neural DNF-MT actor in the overall architecture, a mandatory step of invented predicate discretisation (step 0 in Figure 6.2) is performed at the beginning of the post-training process. The sign of the invented predicate tanh activations is taken, converting them

to ± 1 to interpret them as bivalent logical truth values of $\top$ or $\bot$. Each invented predicate is defined as a minimal set of raw observations using an ASP optimisation function (step 5 in Figure 6.2).

6.2.4 Neural-bivalent-logic translation for Deterministic Policy Interpretation and Policy Intervention

When the neural DNF-MT model encodes a deterministic policy, the translation between the model and its ASP program is bidirectional and maintains truth value equivalence. The key observation is that the bivalent interpretation of the neural DNF-MT model is defined via the tanh output $\hat{\mathbf{y}}$ of the disjunctive layer, not the mutex-tanh output $\tilde{\mathbf{y}}$ used to compute action probabilities during training. The role of $L^{(4)}$ in Equation (6.7) in the training loss is precisely to ensure that this interpretation is sound: by driving the tanh outputs $\hat{\mathbf{y}}$ towards ± 1 for deterministic policies, it guarantees a well-defined bivalent interpretation and makes the thresholding in Step (2.b) valid. Consequently, the neural DNF-MT model is logically interpreted as a neural DNF model with tanh activation in its disjunctive layer, and the bidirectional truth value equivalence established for neural DNF models in Section 4.2 applies directly. The neural-to-bivalent-logic direction of this translation provides interpretability for the learned policy, and the bivalent-to-neural-logic direction enables policy intervention without retraining, as demonstrated in Section 6.3.5.

6.3 Experiments

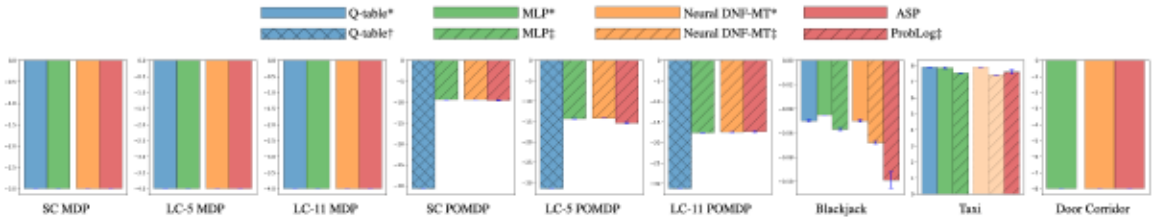


Figure 6.3: Mean episodic return ($\pm$ standard error) across all environments for baseline methods, neural DNF-MT models, and the ProbLog/ASP programs extracted from the trained neural DNF-MT models. Higher values indicate better performance. Q-tables are trained using Q-learning, while MLP and neural DNF-MT actors are trained using actor-critic PPO with MLP critics. In the Taxi environment only, the neural DNF-MT actor is distilled from a pre-trained MLP actor (indicated by dashed borders and faded colouring). Symbols denote action selection methods: * argmax selection, † ϵ-greedy sampling, ‡ sampling from the actor's distribution.

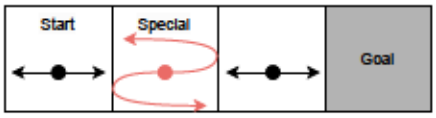
This section evaluates the RL performance (measured in episodic return) of neural DNF-MT actors and their interpreted logical policies in four sets of environments with various forms of observations. Some tasks require stochastic behaviours, while others can be solved with deterministic policies. The evaluation compares the proposed method with two baselines: Q-tables trained with Q-learning where applicable and MLP actors trained with actor-critic PPO. The neural DNF-MT actors are trained with MLP critics using the PPO algorithm in the Switcheroo Corridor set, Blackjack and Door Corridor environments. In the Taxi environment, a neural DNF-MT actor is distilled from a trained MLP actor. The extracted ProbLog policies are not directly evaluated because of the long ProbLog query time. Instead, the evaluation uses their final neural DNF-MT actors before

logical rule extraction (i.e. after step 3, re-pruning) as an approximation. This approximation is close because a ProbLog policy's action distribution is the same as its corresponding neural DNF-MT's action distribution to at least 3 decimal places. A performance evaluation summary is shown in Figure 6.3. The following sections discuss the details of the experiments in each environment.

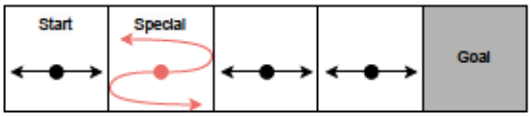
6.3.1 Switcheroo Corridor

The experiments adopt an example environment from [150, Chapter 13.1] and create a set of Switcheroo Corridor environments that support MDP tasks with deterministic policies and POMDP tasks with stochastic policies. The observation can be either (i) a one-hot encoding of the agent's current state number, representing its position (an MDP task), or (ii) the wall status of the agent's current position (a POMDP task). In most states, going left or right results in moving in the intended direction. However, there are special states that reverse the action effect. Thus, the nature of the task decides whether the optimal policy is deterministic or stochastic. In the MDP setting, the optimal policy is deterministic: identifying the special states based on the state number and going left in them. In the POMDP setting, identifying the special states based solely on wall status observations is impossible without a memory. The optimal policy shows stochastic behaviour so that the correct action may be sampled when the agent is in the special states.

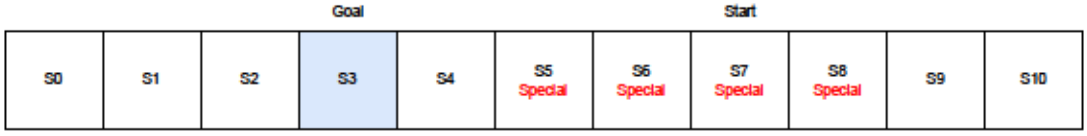
The start, goal, and special states are customisable. Three corridors are created based on different configurations: Small Corridor (SC), as shown in Figure 6.4a, Long Corridor-5 (LC-5), and Long Corridor-11 (LC-11), to test the actor's learning ability when the environment complexity increases. Table 6.1 shows the configurations of the three corridors. Once created, the corridor is fixed throughout training and inference. The reward is set to -1 for each step. The episode is terminated when the agent reaches the goal state or is truncated after the maximum number of steps. By default, the truncation step limit is 50.



(a) Small corridor (SC), same as the one from [150]. The agent starts at state 0 and must reach state 3 (0-indexed). State 2 is the special state that reverses the action effect.



(b) Long Corridor-5 environment, created according to the configuration shown in Table 6.1.



(c) Long Corridor-11 environment, created according to the configuration shown in Table 6.1. Note that S0, S1 and S2 are unreachable states.

Figure 6.4: The three environments in Switcheroo Corridor Environment Set.

Name	Short code	Corridor Length	Start State	Goal State	Special State(s)
Small Corridor	SC	4	0	3	[1]
Long Corridor-5	LC-5	5	0	4	[1]
Long Corridor-11	LC-11	11	7	3	[5, 6, 7, 8]

Table 6.1: Environment configurations for the three environments in the Switcheroo Corridor Environment Set, including SC's configuration.

Environment	Q-Table	MLP	Neural DNF-MT	NDNF-MT: ASP
SC MDP	-3.000 ± 0.000	-3.000 ± 0.000	-3.000 ± 0.000	-3.000 ± 0.000
LC-5 MDP	-4.000 ± 0.000	-4.000 ± 0.000	-4.000 ± 0.000	-4.000 ± 0.000
LC-11 MDP	-4.000 ± 0.000	-4.000 ± 0.000	-4.000 ± 0.000	-4.000 ± 0.000

Table 6.2: Performance of the models in the Switcheroo Corridor MDP environment set, together with the ASP programs extracted from their corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean ± standard error. Argmax action selection is used for all models.

Environment	Q-Table†	MLP‡	Neural DNF-MT‡	NDNF-MT: ProbLog‡
SC POMDP	-30.683 ± 0.006	-9.342 ± 0.022	-9.336 ± 0.019	-9.607 ± 0.129
LC-5 POMDP	-31.459 ± 0.004	-14.304 ± 0.047	-14.212 ± 0.031	-15.358 ± 0.193
LC-11 POMDP	-31.235 ± 0.004	-17.625 ± 0.051	-17.433 ± 0.056	-17.361 ± 0.183

Table 6.3: Performance of the models in the Switcheroo Corridor environment set, together with the ProbLog programs extracted from their corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean ± standard error. Different symbols after the actor's name indicate different action selection methods: † for ϵ-greedy sampling, and ‡ for actor's distribution sampling.

The first six groups in Figure 6.3, Table 6.2, and Table 6.3 show the performance of all models in the environment set. In MDP settings, all methods using argmax action selection perform equally well, reaching the goal with the minimum number of steps. In POMDP settings, MLP and neural DNF-MT actors perform better than Q-table with ϵ-greedy sampling as expected, with minor performance differences. Neural DNF-MT actors provide interpretability via logical programs compared to MLP actors. Listing 6.1 shows the ASP program for a neural DNF-MT actor in SC MDP, where state 1 is identified as special. Listing 6.2 shows the ProbLog rules for a neural DNF-MT actor in SC POMDP. As shown in line 1 in Listing 6.2, the actor favours the action going right when only the left wall is present, which only happens in state 0. Line 2 shows the case when the agent is in either state 1 or 2 with no wall on either side, and the actor shows close to 50-50 preference for both actions. The logical representations for policies learned in LC-5 and LC-11 are provided in Appendix C.2, and they show similar behaviour as the SC MDP and POMDP policies.

```
1 action(left) :- in_s_1.    action(right) :- not in_s_1.
```

Listing 6.1: ASP rules of a neural DNF-MT actor in SC MDP.

```
0.041::action(left) ; 0.959::action(right) :- left_wall_present, \+
    right_wall_present.
0.581::action(left) ; 0.419::action(right) :- \+ left_wall_present,
    \+ right_wall_present.
```

Listing 6.2: ProbLog rules of a neural DNF-MT actor in SC POMDP.

6.3.2 Blackjack

The Blackjack environment[1] from [150, Chapter 5.1] is a simplified version of the Blackjack card game, where the goal is to beat the dealer by having a hand closer to 21 without going over. The agent sees the sum of its hand, the dealer's face-up card, and whether it has a usable ace. It can choose to hit or stick. The environment terminates when the agent sticks or its hand exceeds 21, and truncation never occurs. It receives a reward of +1 for winning, -1 for losing and 0 for drawing. Instead of using Gymnasium's tuple observation (the player's current sum, the dealer's hand, and the player's usable ace), each value in the tuple is converted to a one-hot vector mapped to $\{-1, 1\}$ encoding and stacked together as the observation. There are 44 bits in total in the final observation encoding. The observation input to the neural DNF-MT actor is shown in Figure 6.1a.

Model	Episodic return	Win rate	Policy Divergence
S&B optimal policy [150]*	-0.050 ± 0.001	$42.94\% \pm 0.00\%$	NA
MLP*	$\mathbf{-0.045 \pm 0.000}$	$\mathbf{43.24\% \pm 0.02\%}$	$\mathbf{15.87\% \pm 0.30\%}$
MLP‡	-0.057 ± 0.001	$42.84\% \pm 0.02\%$	
NDNF-MT*	-0.050 ± 0.001	$42.82\% \pm 0.06\%$	$20.66\% \pm 0.56\%$
NDNF-MT‡	-0.068 ± 0.001	$42.17\% \pm 0.03\%$	
ProbLog‡	-0.099 ± 0.007	$40.79\% \pm 0.31\%$	$27.92\% \pm 1.25\%$

Table 6.4: Blackjack: performance of MLP actors, neural DNF-MT actors, and the extracted ProbLog programs. Policy divergence measures the proportion of states where the argmax policy disagrees with the optimal policy from [150].

The performance across the models is shown in the 7^{th} group in Figure 6.3 and Table 6.4. The baseline optimal policy from [150] only shows a single action per state, so it is only evaluated with argmax action selection. The MLP and neural DNF-MT actors are evaluated with both argmax action selection and actor's distribution sampling. MLP actors with argmax action selection perform better than their distribution sampling counterparts, with a higher episodic return and win rate. The same is observed for neural DNF-MT actors. The extracted ProbLog rules[2] perform worse than their original neural DNF-MT actors (no post-training processing), with a higher policy divergence from the optimal policy from [150]. A policy change is also observed from a trained neural DNF-MT actor to its extracted ProbLog policy. Figure 6.5 shows the policy grids of a neural DNF-MT actor trained in the Blackjack environment without any post-training processing, and Figure 6.6 shows its extracted ProbLog policy grid. In both figures, a red square indicates that the argmax action of the agent is different from the baseline optimal policy from [150]. The extracted ProbLog policy

[1] The experiments use the Gymnasium's implementation at https://Gymnasium.farama.org/environments/toy_text/blackjack/.

[2] An extracted ProbLog program example is listed in Appendix C.3.

grid shows more errors (red squares) compared to the original neural DNF-MT actor without any post-training processing. This unwanted policy change caused by thresholding leads to performance loss and persists in later environments; this issue is discussed further in Section 6.4.

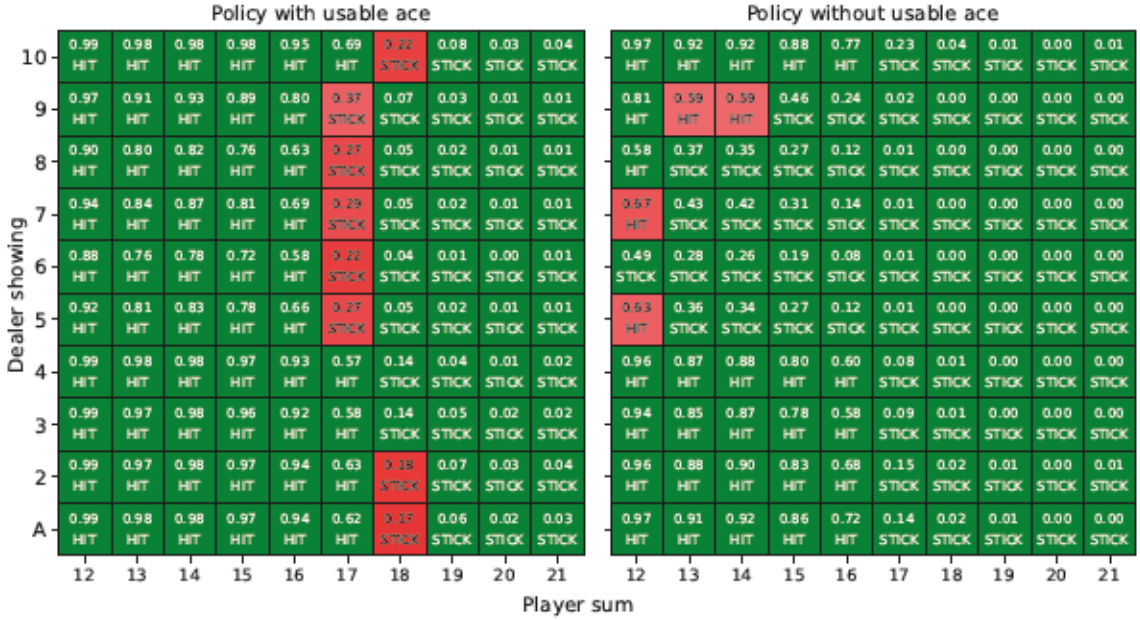


Figure 6.5: Policy grid of a neural DNF-MT actor in the Blackjack environment without post-training processing. Each square shows the probability of taking action 'Hit' and the argmax action in the text. If a square is red, it means that its argmax action is different from the baseline optimal policy from [150]. The redder the square, the larger the probability difference.

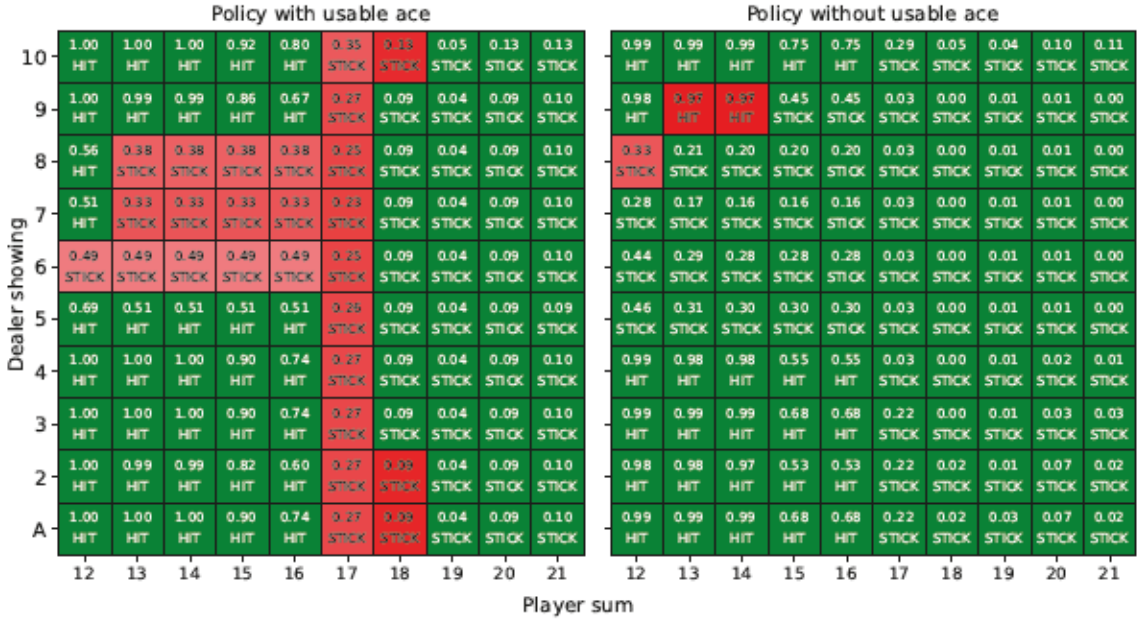


Figure 6.6: Extracted ProbLog policy grid of the same neural DNF-MT actor in Figure 6.5.

6.3.3 Taxi

In the Taxi environment[3] from [41], shown in Figure 6.7, the agent controls a taxi to pick up a passenger first and drop them off at the destination hotel. The environment has 4 coloured squares where a hotel and the passenger will be allocated initially at the start, and the hotel location remains unchanged throughout the episode. A state number is used as the observation, and it encodes the taxi, passenger and hotel locations using the formula $((taxi_row * 5 + taxi_col) * 5 + passenger_loc) * 4 + destination$. Apart from moving in four directions, the agent can pick up/drop off the passenger, but illegally picking up/dropping off will be penalised. Each step gives a -1 reward, but illegally executing 'pickup' and 'drop-off' gives a -10 reward. If a passenger is successfully delivered to the hotel, the agent receives a +20 reward.

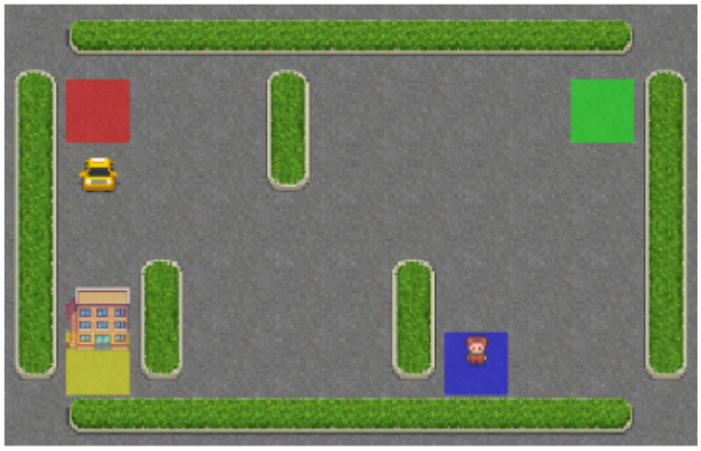


Figure 6.7: Taxi Environment: the taxi needs to pick up a passenger and drop them off at the destination hotel to finish the episode.

The environment is designed for hierarchical reinforcement learning but is solvable with PPO and without task decomposition. However, for both MLP actors and neural DNF-MT actors, the performance is found to be more sensitive to PPO's hyperparameters and fine-tuning the hyperparameters is more difficult than in other environments. The environment is complex due to its hierarchical nature, and learning task dependencies based solely on state numbers proves difficult, as a single-unit change in the taxi's x or y coordinate results in a state number change of 100 or 10. On-policy actor-critic methods can be sensitive to the choice of hyperparameters [10, 45], and with the wrong set of hyperparameters in such a complex environment, the actor easily settles at a local optimum with a reward of -200: never perform 'pickup'/'drop-off' and move until the step limit (200 steps). MLP actors were successfully trained with actor-critic PPO after extensive hyperparameter tuning. However, the combination of the model's structural constraints, the difficulty of the task and the sensitivity of deep RL to hyperparameter choices appears as a barrier to training neural DNF-MT actors and no working set of hyperparameters could be found to train neural DNF-MT actors. The neural DNF-MT model is more sensitive to hyperparameters than an unconstrained MLP due to the architectural constraints required to keep the model interpretable in logical form: the untrainable biases that act as logical biases and the bounded tanh-based activations restrict the

[3]The experiments use Gymnasium's implementation at https://Gymnasium.farama.org/environments/toy_text/taxi/.

optimisation landscape compared to a standard MLP. The reinforcement learning setting appears to amplify this sensitivity relative to supervised learning, as the non-stationary training distribution and the noisy policy-gradient signal interact poorly with these constraints, and small hyperparameter changes can push the actor into the local optimum of avoiding 'pickup'/'drop-off' altogether described above. Therefore, instead of training a neural DNF-MT actor directly, a neural DNF-MT actor is distilled from a trained MLP actor, taking the same observation as input and aiming to output the exact action probabilities as the MLP oracle. The distillation procedure separates policy learning (handled by an MLP oracle) from interpretable policy representation (handled by the neural DNF-MT student), sidestepping the optimisation difficulties of training a constrained model directly with PPO while preserving the interpretability benefits of the neural DNF-MT actor. Other mitigations such as increasing the model capacity or alternative training regimes are also possible. A more systematic investigation of these directions, together with automated hyperparameter tuning tailored to neural DNF-based actors, is left for future work.

Actor Model	Episodic return
Q-table*	7.913 ± 0.009
MLP*	7.865 ± 0.066
MLP‡	7.550 ± 0.011
Neural DNF-MT*	$\mathbf{7.926 \pm 0.009}$
Neural DNF-MT‡	7.424 ± 0.009
NDNF-MT: ProbLog‡	7.604 ± 0.139

Table 6.5: Taxi: performance of Q-tables, MLP actors, neural DNF-MT actors, and the extracted ProbLog programs from the corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean ± standard error. Different symbols after the actor's name indicate different action selection methods: * for argmax action selection, and ‡ for actor's distribution sampling.

The performance is shown in the 8^{th} group in Figure 6.3 and in Table 6.5. Actors using argmax action selection perform better than their distribution sampling counterparts. Again, a performance drop is observed in extracted ProbLog rules.[4] With 300 unique possible starting states, the extracted ProbLog rules are not guaranteed to finish in 200 steps: 2 out of the 10 ProbLog evaluations with action probabilities sampling have truncated episodes. Across ten post-training-processed neural DNF-MT actors with argmax action selection, there are an average of 3.3 unique starting states where the models cannot finish the environment within 200 steps. Decoding the location information encoded in the state number seems to be complicated and makes it hard to learn concise conjunctions, thus increasing the error rate in the post-training processing.

6.3.4 Door Corridor

Inspired by Minigrid [27], a corridor grid with a fixed configuration called Door Corridor is designed, as shown in Figure 6.8. The goal of the agent is to step into the goal cell within a termination limit of 270.

The agent observes a 3×3 grid in front of it with two channels of objects and cell status, and has a choice of four actions: turn left, turn right, move forward, and toggle. The toggle action only

[4] An extracted ProbLog program example is listed in Appendix C.4.

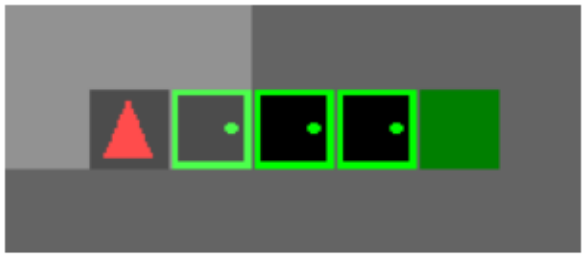


Figure 6.8: Door Corridor (DC): the agent needs to turn right first, and toggle and go through three doors to reach the end of the corridor.

changes the status of a door right in front of the agent. Listing 6.3 shows the action and observation space of the Door Corridor environment.

```
# Action space
class DoorCorridorAction(IntEnum):
    LEFT = 0
    RIGHT = 1
    FORWARD = 2
    TOGGLE = 3

# Observation space, the two channels are object and state
class Object(IntEnum):
    UNSEEN = 0
    EMPTY = 1
    WALL = 2
    DOOR = 3
    AGENT = 4
    GOAL = 5

class State(IntEnum):
    OPEN = 0    # Most objects will be OPEN
    CLOSED = 1  # A door can be OPEN or CLOSED
```

Listing 6.3: Door Corridor Environment action space and observation space.

For this environment, the architecture shown in Figure 6.1b is used, where an encoder is shared between the actor and the critic. The encoder uses a 2D convolutional layer followed by a linear layer to invent predicates in GBL form. The performance of MLP actors, neural DNF-MT actors and their extracted ASP programs are shown in the last group in Figure 6.3 and Table 6.6. Both of the neural actors learn the optimal deterministic policy.

To evaluate an extracted ASP program in the environment, the 3×3 observation is first passed to the encoder, invented predicates with bivalent interpretations $\top$ are converted to ASP facts, and then these facts are appended as context to the base policy. The combined ASP program has to (i) output one stable model with only one action at each step (logically mutually exclusive) and (ii) finish without truncation to be counted as a successful run. These requirements are also reflected in

Actor Model	Episodic return
MLP	-8.000 ± 0.000
Neural DNF-MT	-8.000 ± 0.000
NDNF-MT: ASP	-8.000 ± 0.000

Table 6.6: Door Corridor: performance of MLP actors, neural DNF-MT actors, and the extracted ASP programs from the corresponding neural DNF-MT actors. The episodic return metric is represented in the form of mean ± standard error. Argmax action selection is used for all models.

the neural DNF-MT actor: the final tanh activation should only have one value greater than 0, and taking that only action with greater-than-0 tanh activation at each step finishes the environment without truncation. The auxiliary loss terms in Equations 6.4, 6.6 and 6.7 help the neural DNF-MT actor to achieve these requirements but make the training less likely to converge on a good solution. Out of 32 runs, 6 runs cannot finish the environment within the step limit. However, 25 of the 26 remaining runs can be interpreted as ASP policies. For the single failing case, it fails to maintain logical mutual exclusivity after thresholding. While it is possible to extract a ProbLog program from it, the environment is known to support an optimal deterministic policy. Hence, no logical program is extracted for this run. The ASP programs of the 25 runs successfully finish the environment with minimal steps, as reported in Figure 6.3. Listing 6.4 shows an example of the extracted ASP program from one of the successful runs and a possible set of definitions for the invented predicates.

```
action(turn_right) :- a_5, a_8.
action(forward) :- a_2.
action(toggle) :- a_3.
% Definitions of each invented predicate a_i:
a_2 :- top_right_corner_wall.
a_3 :- one_step_ahead_closed_door.
a_5 :- not curr_location_open_door,
       not one_step_ahead_closed_door.
a_8 :- two_step_ahead_unseen.
```

Listing 6.4: An ASP policy for a neural DNF-MT actor in DC, with a possible set of definitions for the invented predicates.

The invented predicates are extracted via an ASP optimisation algorithm. Given the trace of an agent that completes the environment without truncation, all experienced observations are first translated into ASP facts. For example, coordinate (2, 2) (i.e. the cell to the right of the agent) being a closed door is represented as `obs(2, 2, 3, 1)`, where the first two elements denote the cell coordinates, the third element indicates the object type, and the fourth element specifies the state, as defined in Listing 6.3. Each observation is also associated with its corresponding timestamp; for instance, `fired_obs_at_timestep(0, obs(2, 2, 3, 1))` indicates that `obs(2, 2, 3, 1)` occurred at timestamp 0. Similarly, each element of the image encoding vector (or the invented predicates), which serves as input to the neural DNF-MT actor, is translated into an ASP fact, such as `img_encoding_id(8)` for the 8^{th} element. The activation status of these elements at each timestamp is also recorded; for example, `fired_img_encoding_gt(1, a(8))` indicates that the 8^{th} element was activated at timestamp 1. The objective of the ASP optimisation is to, for

each element in the image encoding vector (invented predicate), identify a minimal set of positive or negated observations, possibly empty, that uniquely triggers this element (invented predicate). This set of observations then forms the body of a rule with the image encoding element as the head, thereby providing an interpretation of the invented predicates. When the optimisation succeeds, multiple stable models may exist, each offering a different interpretation. However, a solution is not guaranteed; if no stable model is found, the invented predicates remain uninterpretable.

6.3.5 Policy Intervention in Door Corridor Variants

Two variations of the base Door Corridor environment are created with only different termination conditions: Door Corridor-T (DC-T), where the agent must be in front of the goal and toggle it instead of moving into it; and Door Corridor-OT (DC-OT), where the agent must stand on the goal and take the action 'toggle'. Table 6.7 shows the optimal policy's actions for the Door Corridor environment and its variants.

Environment	Optimal policy's actions (text)	Optimal policy's actions (int)
DC	[Right, Toggle, Forward, Toggle, Forward, Toggle, Forward, Forward]	[1, 3, 2, 3, 2, 3, 2, 2]
DC-T	[Right, Toggle, Forward, Toggle, Forward, Toggle, Forward, Toggle]	[1, 3, 2, 3, 2, 3, 2, 3]
DC-OT	[Right, Toggle, Forward, Toggle, Forward, Toggle, Forward, Forward, Toggle]	[1, 3, 2, 3, 2, 3, 2, 2, 3]

Table 6.7: Optimal policy's actions for the Door Corridor (DC) environment and its variants, with blue text showing the difference in the variants. The difference in the optimal policy's actions is highlighted in blue.

The input observation remains unchanged since only the goal cell's mechanism changes. The encoder can be reused immediately, but the actor and critic cannot. An MLP actor trained on DC fails to finish within step limits in DC-T and DC-OT environments without re-training. However, the ASP policy can be modified to achieve the optimal reward in both DC-T and DC-OT environments. Listings 6.5 and 6.6 show the modified ASP programs for DC-T and DC-OT environments, respectively. The modified ASP programs can be ported back to neural DNF-MT actors by virtue of the bidirectional neural-bivalent-logic translation. The modified neural DNF-MT actors also finish DC-T and DC-OT environments with minimal steps without any re-training.

```
action(turn_right) :- a_5, a_8.
action(forward) :- not a_1, a_2.
action(toggle) :- a_3.
action(toggle) :- a_1, not a_3, a_12.
```

Listing 6.5: Policy for DC-T, modified from Listing 6.4's policy.

```
action(turn_right) :- a_5, a_8, a_11.
action(forward) :- a_2.
action(toggle) :- a_3.
action(toggle) :- not a_2, not a_3, not a_11.
```

Listing 6.6: Policy for DC-OT, modified from Listing 6.4's policy.

6.4 Discussions

This section first analyses the persistent performance loss issue in Blackjack, Taxi, and Door Corridor environments. The discussion then covers the implications of this work in terms of performance, interpretability, inference, and policy intervention.

6.4.1 Performance Loss due to Thresholding

The thresholding step converts the target layer(s) from a weighted continuous space to a discrete space with only 0 and ± 6 values, saturating the tanh activation at ± 1 and enabling the translation to bivalent logic. However, the thresholding step may not maintain the same logical interpretation of the layer output. This issue is shown through an example in Listing 6.7, where a thresholded neural DNF-MT actor fails to maintain logical mutual exclusivity in the Door Corridor environment. Note that thresholding is applied on both the conjunctive and disjunctive layers since a deterministic policy is desired.

```
% Conjunctive nodes and their learned weights:
c_0  =  3.03 x_7  + bias_c0
c_7  =  0.56 x_13 + bias_c7
c_9  = -1.56 x_2  + bias_c9
c_11 = -1.05 x_9  + bias_c11
% Disjunctive nodes and their learned weights:
d_1  =  4.58 c_0 + bias_d1
d_2  = -3.48 c_9 + bias_d2
d_3  =  1.29 c_7 + 0.76 c_9 + 4.33 c_11 + bias_d3
% Thresholded nodes with tau = 0 (and ASP translation):
c_0  =  6 x_7                  (conj_0  :- a_7.)
c_7  =  6 x_13                 (conj_7  :- a_13.)
c_9  = -6 x_2                  (conj_9  :- not a_2.)
c_11 = -6 x_9                  (conj_11 :- not a_9.)
d_1  =  6 c_0                  (act(1)  :- conj_0.)
d_2  = -6 c_9                  (act(2)  :- not conj_9.)
d_3  =  6 c_7 + 6 c_9 + 6 c_11 + 12
(act(3)  :- conj_7. act(3) :- conj_9. act(3) :- conj_11.)
```

Listing 6.7: A neural DNF-MT actor in the Door Corridor environment that fails at the thresholding stage. The bias terms are left uncalculated for brevity.

The 1st row of values in Table 6.8 are the pre-thresholding tanh output when $x_2 = -1, x_7 = 1, x_9 = 1, x_{13} = -1$, with only $\hat{y_1}$ interpreted as $\top$ and chosen as action. The thresholded conjunctive nodes in the 2nd row share the same sign as row 1, but $\hat{y_3}$ becomes positive after thresholding,

resulting in two actions being $\top$ and thus violating logical mutual exclusivity. The disjunctive nodes' original weights achieve the balance of importance between c_7, c_9 and c_{11} to make $\hat{y_3}$ negative. However, the thresholding process ignores the weights and makes them equally important, leading to a different output and truth value. It shows that the thresholding stage cannot handle volatile and interdependent weights and maintain the underlying truth table represented by the model. This issue is further discussed with a proposed solution in Chapter 7.

c_0	c_7	c_9	c_{11}	$\hat{y_1}$	$\hat{y_2}$	$\hat{y_3}$
1.00	-0.51	0.92	-0.78	1.00	-1.00	-0.86
1.00	-1.00	1.00	-1.00	1.00	-1.00	1.00

Table 6.8: Conjunctive and disjunctive nodes' tanh output when $x_2 = -1, x_7 = 1, x_9 = 1, x_{13} = -1$, calculated based on the formulation in Listing 6.7. Row 1 is the original output without applying thresholding, and row 2 is the output after thresholding on value 0.

6.4.2 Implications of Neural DNF-MT Model

Performance

The experiments show that the neural DNF-MT actor can be trained with actor-critic PPO or distilled from an MLP oracle to learn an optimal policy in the Switcheroo Corridor, Taxi, and Door Corridor environments. In Blackjack, the performance is worse but close to an optimal MLP actor. Furthermore, the results demonstrate that an encoder for handling complex observations and realising predicate invention can be end-to-end trained with the neural DNF-MT actor in the Door Corridor environment.

Interpretability

The logical programs extracted from trained neural DNF-MT actors provide interpretability, which MLP actors lack. The results also demonstrate through different environments that stochastic and deterministic policies can be represented in different forms of logic (ProbLog and ASP, respectively).

Inference

Even when the actor has learnt an interpretable policy, running a fully logic-based agent may not be efficient. Inference in neural DNF-MT actors is significantly faster than in ProbLog or ASP, thanks to tensor operations and environment parallelism. This claim is further supported by runtime comparison experiments. Table 6.9 shows the runtime comparison between different models across various environments. All entries are run on a 10-core Apple M1 Pro CPU with 16G RAM. Experiments with Q-table are implemented in numpy [69], while MLP and neural DNF-MT are implemented in PyTorch [11]. The clingo solver [126] is used to run ASP programs and the ProbLog Python API to perform ProbLog inference. The results show that inference with logic-based models is significantly slower than with neural models.

An additional experiment approximates the time taken to perform inference in ProbLog when the program is large. All programs are in the following form:

Environment	Model	No. episodes	Overall runtime (s)	Avg. runtime (s) per episode
Blackjack	Q-table	1e4	0.416	4.156e−5
	MLP		2.283	2.283e−4
	MLP*		1.157	1.157e−4
	NDNF-MT		5.564	5.564e−4
	NDNF-MT*		1.574	1.574e−4
	ProbLog	10	21.583	2.158
Taxi	Q-table	1e4	1.202	1.202e−4
	MLP		17.338	1.734e−3
	MLP*		7.279	7.279e−4
	NDNF-MT		49.954	4.995e−3
	NDNF-MT*		9.790	9.790e−4
	ProbLog	1	Timed out after 30 min	
Door Corridor	MLP	1e4	10.178	1.018e−3
	MLP*		0.073	7.258e−6
	NDNF-MT		36.726	3.673e−3
	NDNF-MT*		0.111	1.112e−5
	ASP	1000	25.296	2.530e−2

Table 6.9: Runtime comparison between different models across various environments. The overall runtime is the total time to run the model for the specified number of episodes. Models without * are run in a single environment in a loop for n times. Models with * are run in 8 parallel environments for a total of n episodes.

```
p_0_1 :: action(0) ; ... ; p_0_n :: action(n) :- rule(0).
p_1_1 :: action(0) ; ... ; p_1_n :: action(n) :- rule(1).
...
p_m_1 :: action(0) ; ... ; p_m_n :: action(n) :- rule(m).
```

The experiment uses $n \in \{2, 6\}, m \in \{1..18\}$. The time taken is shown in Figure 6.9, and the results show the overall trend of exponential growth in time to the number of rules. This is only for approximation since the actual policy does not resemble the format used here. A Problog program extracted from a neural DNF-MT actor in a Blackjack environment has 58 annotated disjunctions. It takes 2652.212s to perform 380 inference queries, averaging 7s per query. Regardless, the conclusion remains that inference with ProbLog is more expensive than with neural models.

Policy Intervention

The bidirectional neural-bivalent-logic translation allows modification of the ASP program and translation back to the neural architecture without re-training, as shown in Door Corridor's variations in Section 6.3.5. This feature could be helpful in tasks where background knowledge is available. By pre-encoding the information into logical rules or modifying the logical rules of an actor trained in a similar environment, the edited logic program can be ported back to the neural model to provide a hot start in training. This functionality may be explored in future work.

In summary, the neural DNF-MT model learns interpretable and editable policies, with the neural benefits of end-to-end training and parallelism in inference and the logical benefits of interpretable

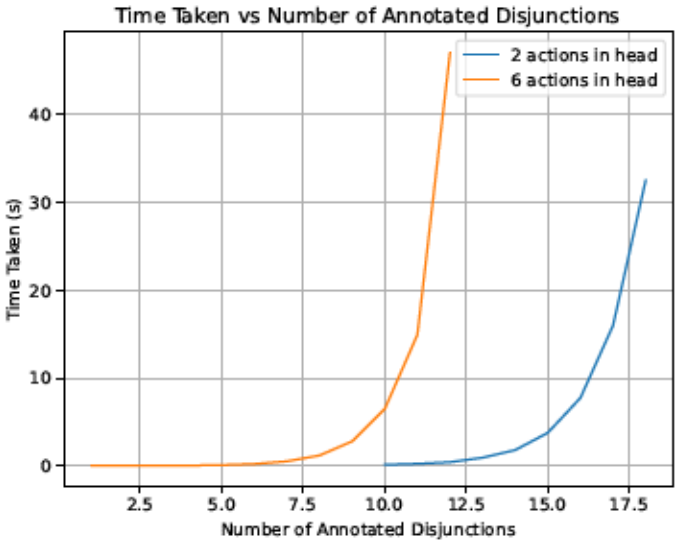


Figure 6.9: Time taken vs the number of annotated disjunctions in ProbLog, using a ProbLog program in a certain format as an approximation. The overall trend is that the time taken increases exponentially with the number of rules presented in the ProbLog program. However, this pattern cannot be confirmed as the same for inference with an actual policy extracted from a neural DNF-MT actor.

logical program representation.

6.5 Conclusion

This chapter proposes a neuro-symbolic approach named the neural DNF-MT model for learning interpretable and editable policies in RL. It can be trained with actor-critic PPO or distilled from a trained MLP actor, and an encoder for predicate invention can also be end-to-end trained together with it. The trained neural DNF-MT model can be represented as either a ProbLog program for a stochastic policy or an ASP program for a deterministic policy. The neural-bivalent-logic translation is bidirectional, allowing policy intervention by modifying the ASP program and then converting it back to the neural model for efficient inference in parallel environments. The neural DNF-MT model is evaluated in four environments with different forms of observations and stochastic/deterministic behaviours. The experiments show the neural DNF-MT model's capability to learn the optimal policy with performance similar to an MLP actor's. Furthermore, it provides logical representation and enables policy intervention, neither of which an MLP can easily provide. Compared to existing neuro-symbolic approaches, such as NLRL [77] and NUDGE [39], the neural DNF-MT model does not use rule templates or mode declarations. Therefore, it does not rely on human engineering to construct the inductive bias and can learn a wider range of rules. In contrast to works that either operate on relational-based observations [77, 39, 170] or require pre-trained networks to extract logical predicates [39, 84], the experimental results demonstrate that the neural DNF-MT model is end-to-end trainable with preceding layers for predicate invention.

Chapter 7

The Entanglement Problem and Disentanglement

Previous chapters introduced two neural DNF-based models, aiming to solve different types of learning tasks: neural DNF-EO for *deterministic* multi-class classification tasks and neural DNF-MT for *probabilistic* policy learning in reinforcement learning settings. These chapters demonstrate the following properties of the neural DNF-based models:

- **Versatility**: the neural DNF-based framework supports diverse learning settings through architectural customisation. The semi-symbolic nodes can be configured to create removable 'constraint' layers (as in the neural DNF-EO model) and their activation functions can be tailored to enforce domain-specific logical constraints (as in the neural DNF-MT model).

- **Usability beyond synthetic settings**: the experiments involve real-world classification tasks with large numbers of binary inputs and multi-class or multi-label outputs (Section 5.2), as well as various reinforcement learning environments (Section 6.3).

- **Interpretability**: a neural DNF or a neural DNF-EO can be translated into a set of bivalent logic rules, and a neural DNF-MT can be translated into a set of probabilistic logical rules. The logical rules can be used to explain the model's behaviour.

- **End-to-end trainability and integration**: the neural DNF-based models remain to be trained end-to-end even when they are integrated with other neural components. Section 6.3.3 and Section 6.3.4 demonstrate the integration of the neural DNF-based models with other neural components such as MLPs and CNNs.

- **Flexibility and scalability**: the neural DNF-based models' architectural similarity to the traditional neural networks makes it easier to bootstrap without the need for background knowledge of the task and able to approximate a large search space. While the use of background knowledge in ILP methods is a merit, the hypothesis search space can be too large for the model to handle when prior knowledge is unavailable, and if the background knowledge is incorrect the optimal solution may be excluded from the search space.

However, the persistent observation of neural DNF-based models' performance degradation at the thresholding step shows that the process fails to fully maintain the models' behaviour due to some fundamental yet uninvestigated issues. This chapter (1) first identifies that the existing thresholding discretisation method fails to faithfully translate some neural units into logical form, causing a performance drop, characterising such neural units as *entangled*; (2) and proposes a disentanglement method to address this issue, better preserving model performance in the extracted symbolic representation. Apart from exploring and solving the entanglement problem, this chapter also (3) presents a simple threshold-learning method for real-valued features: these learned bounds are of the form '*feature* $>$ *learned threshold value*' and act as interpretable invented predicates, and (4) proposes an improved delta scheduling method to reduce the destructive effect of the logical bias on the learning of the neural DNF-based models that leads to poor performance. The experiments demonstrate that the proposed disentanglement method drastically decreases the performance degradation when translating trained neural DNF-based models across binary, multi-class, and multi-label classification tasks (some of which require predicate invention).

The rest of this chapter includes material from our paper [17].

7.1 The Problem of Entanglement

The thresholding discretisation method prepares the node to be translated into a single rule following the translation process in Section 4.2.2. To encourage the weights to be in $\{-6, 0, 6\}^N$, the auxiliary loss function $|w||6 - |w||$ is used during training. However, due to the flexibility of the neural architecture, a semi-symbolic node is more likely to learn a representation that cannot be translated into a single logical formula. Rather, the node's behaviour needs to be described by a set of rules with the same rule head, showing a phenomenon termed *entanglement*. Some weights in the nodes are trained to have absolute values closer to 0 than 6, representing that their corresponding inputs do not always contribute to the node's behaviour and do not appear in all rules but only in some rules in the logical representation. The thresholding method is not able to 'split' the weights but only results in a single rule that cannot fully represent the same behaviour as a set of rules, eventually altering the node's behaviour. Despite having the auxiliary loss function $|w||6 - |w||$ during training to aid the thresholding process, the weights in the semi-symbolic nodes are not guaranteed to be close to ± 6 but take non-zero values between -6 and 6. This section demonstrates this phenomenon in two examples where the thresholding method either underemphasises or overemphasises the entangled weights with smaller magnitudes.

7.1.1 Underemphasising

Example 7.1.1 shows a case when the thresholding method underemphasises the weights of a conjunctive node. The thresholding discretisation that best preserves the node's behaviour (i.e. maintaining the model performance as much as possible) uses any $\tau > 1$, and the resulting weights are $[6, 0, 0]$. The discretised node is translated into $\mathrm{c} \leftarrow \mathrm{a}_1$, but this rule over-generalises. To see this, note that the node is not interpreted as true when $x_1 = 1, x_2 = -1$ and $x_3 = -1$, but the translated rule will

fire in this case. A set of two rules that can perfectly describe the node's behaviour is:

$$\mathfrak{L} = \left\{ \begin{array}{l} \mathrm{c} \leftarrow \mathrm{a}_1, \mathrm{a}_2. \\ \mathrm{c} \leftarrow \mathrm{a}_1, \mathrm{a}_3. \end{array} \right\}.$$

Example 7.1.1: Conjunctive Node with Entangled Representation

Consider a trained conjunctive semi-symbolic node with weights $\mathbf{w} = [3, 1, 1]$. Its bias is calculated as $\beta = \max|\mathbf{w}| - \sum|\mathbf{w}| = -2$. After applying thresholding discretisation with $\tau = 1$, the discretised weights are $[6, 0, 0]$ and $\beta' = 0$. A soft-valued truth table of this node (before and after thresholding) is shown below, with the difference caused by thresholding highlighted in red:

x_1	x_2	x_3	Node activation $\tanh(\sum_{i=1}^{3} w_i x_i + \beta)$	Bivalent Interpretation b	Thresholded node activation $\tanh(6 \cdot x_1 + 0 \cdot x_2 + 0 \cdot x_3 + 0)$	Thresholded bivalent interpretation b'
1	1	1	0.995	$\top$	1	$\top$
1	1	-1	0.762	$\top$	1	$\top$
1	-1	1	0.762	$\top$	1	$\top$
1	-1	-1	-0.762	$\bot$	1	$\top$
-1	1	1	-0.995	$\bot$	-1	$\bot$
-1	1	-1	-0.999	$\bot$	-1	$\bot$
-1	-1	1	-0.999	$\bot$	-1	$\bot$
-1	-1	-1	-0.999	$\bot$	-1	$\bot$

7.1.2 Overemphasising

Example 4.1.1 also shows a conjunctive node with entangled weights. Recall the conjunctive node has weights $[-6, -2, -2, 2, -6]$. The discretisation that best preserves the node's behaviour uses $\tau = 0$, resulting in a discretised conjunctive node with weights $[-6, -6, -6, 6, -6]$ and is translated into: $\mathrm{c} \leftarrow \texttt{not}\ \mathrm{a}_1,\ \texttt{not}\ \mathrm{a}_2,\ \texttt{not}\ \mathrm{a}_3,\ \mathrm{a}_4,\ \texttt{not}\ \mathrm{a}_5$. Although this is the best translation, it still alters the behaviour slightly: rule head c cannot be true when a_4 is not true, but it is possible for the original node's activation to be interpreted as true when $x_4 = -1$. This is because the thresholding method overemphasises the smaller magnitude weights w_2, w_3, and w_4. In fact, the behaviour can be perfectly described by a set of three rules:

$$\mathfrak{L} = \left\{ \begin{array}{l} \mathrm{c} \leftarrow \texttt{not}\ \mathrm{a}_1,\ \texttt{not}\ \mathrm{a}_3,\ \mathrm{a}_4,\ \texttt{not}\ \mathrm{a}_5. \\ \mathrm{c} \leftarrow \texttt{not}\ \mathrm{a}_1,\ \texttt{not}\ \mathrm{a}_2,\ \texttt{not}\ \mathrm{a}_3,\ \texttt{not}\ \mathrm{a}_5. \\ \mathrm{c} \leftarrow \texttt{not}\ \mathrm{a}_1,\ \texttt{not}\ \mathrm{a}_2,\ \mathrm{a}_4,\ \texttt{not}\ \mathrm{a}_5. \end{array} \right\}. \tag{7.1}$$

By the lower-weighted terms only appearing in subsets of the rule bodies, it reflects the smaller influence of x_2, x_3, x_4 on the node's behaviour.

7.2 Exploration of Stronger Logical Bias: `Semi-Symbolic-MinBias` Node

As discussed, the weights in a semi-symbolic node encode a form of 'importance', reflecting how much each input contributes to the node's belief. This property allows a single semi-symbolic node

to represent multiple rules with the same head as a soft rule, resulting in entangled representations. Rather than addressing this entanglement through post-training modifications, an alternative perspective is to view it as a violation of the node's intended logical semantics. This section argues that the root cause of such violations lies in the formulation of the semi-symbolic node's bias term, which does not enforce a strict bivalent-logical constraint. It then explores an alternative approach that modifies the semi-symbolic nodes themselves by introducing a min version of the bias, designed to produce learned representations that adhere more closely to bivalent logic and therefore require no post-training adjustments. However, preliminary experiments reveal that training with the min bias is considerably more difficult than with the max version, rendering this approach impractical for real-world applications.

7.2.1 `Semi-Symbolic-MinBias`, a Stricter Logical Constraint

Max bias acts as a soft logical constraint. Consider again the conjunctive node from Example 7.1.1 with weights $\mathbf{w} = [3, 1, 1]$. The raw output of this node for the input $\mathbf{x} = [1, -1, 1]$ is computed as:

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) =& 3 \cdot 1 + 1 \cdot (-1) + 1 \cdot 1 + \beta \\ =& 3 - 1 + 1 + (\max|\mathbf{w}| - \sum|\mathbf{w}|) \\ =& 3 - 1 + 1 + (3 - (3 + 1 + 1)) \\ =& 1 \end{aligned}$$

The positive output indicates that the node evaluates to true. However, under a stricter semi-symbolic constraint adhering to bivalent logic, the node should instead evaluate to false: one of the inputs is false, and a bivalent conjunction returns false whenever any conjunct is false. Consequently, the current semi-symbolic node exhibits behaviour inconsistent with bivalent logic. Since the logical constraint is enforced through the computed bias term rather than the learnable weights, modifying the bias formulation offers a pathway to impose stricter bivalent constraints on the node's behaviour.

Derivation of a stricter logical constraint encoded in the bias term. The derivation of a stricter bias formulation follows a similar approach to that used for the semi-symbolic node in pix2rule [28]. Consider a conjunctive semi-symbolic node with N inputs, where each x_i is strictly ± 1. The definition of a conjunction comprises two conditions: (1) if all conjuncts are true, the conjunction is true; and (2) if any conjunct is false, the conjunction is false. These conditions can be expressed for the semi-symbolic node as follows:

$$\sum_{i}^{N} |w_i| + \beta = z \qquad \text{All conjuncts are true} \tag{7.2}$$

$$\sum_{i \neq j}^{N} |w_i| - |w_j| + \beta = -z \qquad \text{Conjunct } j \text{ is false} \tag{7.3}$$

where z is positive. Equation (7.3) represents the case where exactly one conjunct, indexed by j, is

false. Combining Equations (7.2) and (7.3) yields:

$$
\begin{aligned}
-2\beta &= \sum_{i}^{N} |w_i| + \sum_{i,i\neq j}^{N} |w_i| - |w_j| \\
-2\beta &= 2\sum_{i}^{N} |w_i| - 2|w_j| \\
\beta &= |w_j| - \sum_{i}^{N} |w_i|
\end{aligned}
$$

This result provides a starting point for determining an appropriate value of β. When multiple conjuncts are false, the conjunction must remain false, requiring the output to be negative. Letting $\beta = \alpha - \sum_{i}^{N} |w_i|$, the node's output is calculated as follows:

$$
\begin{aligned}
\sum_{i \text{ s.t. } w_i x_i > 0}^{N} |w_i| - \sum_{j \text{ s.t. } w_j x_j < 0}^{N} |w_j| + \beta < 0 \\
\sum_{i \text{ s.t. } w_i x_i > 0}^{N} |w_i| - \sum_{j \text{ s.t. } w_j x_j < 0}^{N} |w_j| + \alpha - \sum_{i}^{N} |w_i| < 0 \\
\alpha - 2 \sum_{j \text{ s.t. } w_j x_j < 0} |w_j| < 0 \\
\alpha < 2 \sum_{j \text{ s.t. } w_j x_j < 0} |w_j|
\end{aligned}
\tag{7.4}
$$

The minimal case satisfying Condition (2) occurs when exactly one conjunct is false. In this scenario, Inequality (7.4) simplifies to:

$$\alpha < 2|w_j| \tag{7.5}$$

where j is the index of the single false conjunct.

A suitable choice for α satisfying Inequality (7.5) is $\min_{i,w_i\neq 0} |w_i|$. Even if $\min_{i,w_i\neq 0} |w_i|$ equals $|w_j|$, the inequality $|w_j| < 2|w_j|$ remains satisfied. When $w_j = 0$, the situation reduces to Equation (7.2); since the output takes the form $\alpha - 2\sum_{j \text{ s.t. } w_j x_j < 0} |w_j|$, it remains positive. Substituting $\alpha = \min_{i,w_i\neq 0} |w_i|$ into Inequality (7.4) ensures the inequality holds in general: the base case of a single false conjunct has been verified, and additional false conjuncts only increase the right-hand side, preserving the inequality. In conclusion, a conjunctive node with a stricter bivalent logic constraint employs the following bias:

$$\beta_{\min} = \min_{i,w_i\neq 0} |w_i| - \sum_{i}^{N} |w_i| \tag{7.6}$$

Unlike the max version of the bias, $\beta_{\max} = \max_i |w_i| - \sum_{i}^{N} |w_i|$, which allows intermediate weight magnitudes to encode varying degrees of importance, the min version disregards the scale of individual weights. Note that the choice between max and min does not affect the logical semantics when all weights are strictly ± 6.

Applying the min bias to the node with weights $\mathbf{w} = [3, 1, 1]$ from Example 7.1.1 produces the

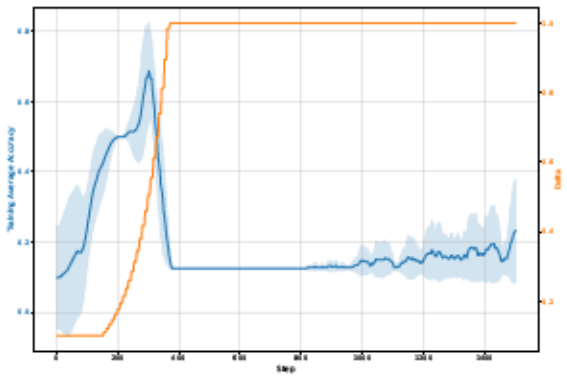
expected behaviour:

$$\begin{aligned} & 3 \cdot 1 + 1 \cdot (-1) + 1 \cdot 1 + \beta_{\text{min}} \\ =& 3 - 1 + 1 + (1 - (3 + 1 + 1)) \\ =& -1 < 0 \end{aligned}$$

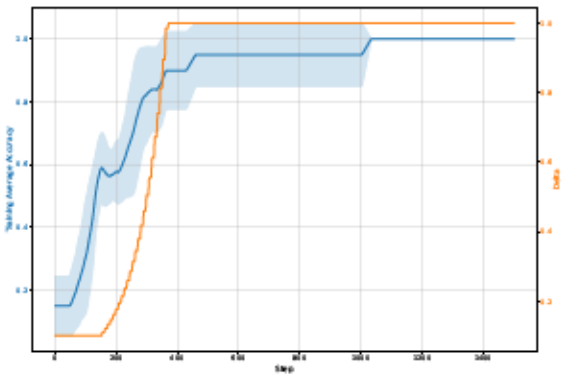
7.2.2 Limitations of `Semi-Symbolic-MinBias`

Although the min bias offers a stricter constraint more closely aligned with bivalent logic, this additional rigidity substantially increases training difficulty. A synthetic experiment demonstrates this challenge. The experiment uses a multi-label classification task with three ground truth ASP rules: {`t0` :– `c`. `t1` :– `a`. `t2` :– `a,b`.}. Two neural DNF models are trained on this task: one employing `Semi-Symbolic-MinBias` nodes and the other using the original semi-symbolic nodes with the max bias. Both models share an identical architecture comprising 9 conjunctive nodes and 3 disjunctive nodes, and each configuration is trained 5 times with different random seeds.

Figures 7.1a and 7.1b present the training accuracy and δ magnitude for both model variants. The results reveal that models with `Semi-Symbolic-MinBias` nodes exhibit performance degradation as δ approaches 1 and ultimately fail to converge to satisfactory solutions. In contrast, models with the original semi-symbolic nodes (max bias) consistently converge to good solutions across all runs. Despite the theoretical appeal of stronger logical constraints, the `Semi-Symbolic-MinBias` formulation proves impractical due to these training difficulties. Developing an improved training regime that accommodates the min bias remains an open problem for future investigation, with candidates such as constrained optimisation and gradient-free learning further discussed in Section 8.2.1



(a) Training accuracy and magnitude of δ of a neural DNF model with `Semi-Symbolic-MinBias` nodes. The performance degradation starts around $|\delta| > 0.6$.



(b) Training accuracy and magnitude of δ of a neural DNF model with original semi-symbolic nodes (max bias). The model converges to a good solution across all runs.

Given that modifying the semi-symbolic nodes directly through the min bias proves impractical, an alternative approach to address the entanglement problem warrants consideration. The next section explores modifications to the post-training process, specifically targeting the thresholding step.

7.3 Disentanglement

This section proposes a method that **disentangles** the weights of semi-symbolic nodes while preserving the nodes' behaviour. The approach transforms a real-valued conjunctive node into multiple smaller discretised nodes that jointly represent the same behaviour as the original node's. The following proposition formalises this idea:

Proposition 7.3.1. *Given a conjunctive node c parameterised by $w \in \mathbb{R}^s$ and taking soft inputs $\mathbf{x} \in [-1,1]^s, s \in \mathbb{N}_{>0}$, there exists a collection of $I \in \mathbb{N}_{>0}$ conjunctive nodes $\{c_i\}_{i=1}^I$ with associated (split) weights $\{\tilde{\mathbf{w}}_i\}_{i=1}^I = \mathcal{W}$, where each $\tilde{\mathbf{w}}_i$ is discretised, i.e. $\tilde{\mathbf{w}}_i \in \{-6,0,6\}^s$, such that under discrete input $\tilde{\mathbf{x}} \in \{-1,1\}^s$ the disjunction of their bivalent interpretations is the same as that of c.*

The following subsections prove Proposition 7.3.1 by constructing a valid set of split weights $\mathcal{W}$ for a conjunctive node c, based on whether the goal is to preserve node c's positive (Section 7.3.1) or negative (Section 7.3.2) behaviour. If c is positively used in a disjunctive node (i.e. the connection weight is positive), the split nodes should jointly encode the positive behaviour of the original node. If c is negatively used in a disjunctive node, (i.e. the connection weight is negative), it means that the negated behaviour of c is used in the disjunctive node. Thus, the split nodes should jointly encode the negative behaviour of c and the sign of the connection weight to the disjunctive node is flipped.

Using Proposition 7.3.1, the full disentanglement discretisation process of a neural DNF-based model can be constructed, which consists of the following three steps (Figure 7.2): (1) split all conjunctive nodes into smaller disentangled and discretised conjunctive nodes; (2) replace the original conjunctive node with the split conjunctive nodes by connecting them to the disjunctive node with the same weight as the original node; and (3) if the model is not a neural DNF-MT model, apply the thresholding discretisation method on the disjunctive layer (either on 0 or sweeping through all possible values). The method only disentangles conjunctive nodes while still applying thresholding to discretise the disjunctive layer because:

1. Split conjunctive nodes can be reconnected to the neural DNF-based model without requiring auxiliary predicates, as the rewiring preserves logical equivalence. Specifically, when a conjunctive node c, connected to a disjunctive node d with a positive weight, splits into $c_1, \ldots, c_k$, the equivalence $(c_1 \vee \ldots \vee c_k) \rightarrow d \equiv (c_1 \rightarrow d) \wedge \ldots \wedge (c_k \rightarrow d)$ guarantees that each split node can be independently connected to d while maintaining the original logical semantics.

2. The splitting process based on Proposition 7.3.1 requires binary inputs in $\{-1,1\}^N$. While conjunctive layer inputs are typically binary, disjunctive layer inputs cannot be guaranteed to be binary, even with auxiliary loss functions. Thus, the splitting process can only be applied to conjunctive nodes and not to disjunctive nodes.

7.3.1 Disentangling a Positively Connected Conjunctive Node

This subsection provides some intuition for Proposition 7.3.1 and then gives a proof sketch. The complete proof is given in Appendix D.

Given a single conjunctive node, c, parameterised by $\mathbf{w} \in \mathbb{R}^s$ with a total of s inputs and connected to a disjunctive node with a positive weight, the goal is to find a collection of split nodes

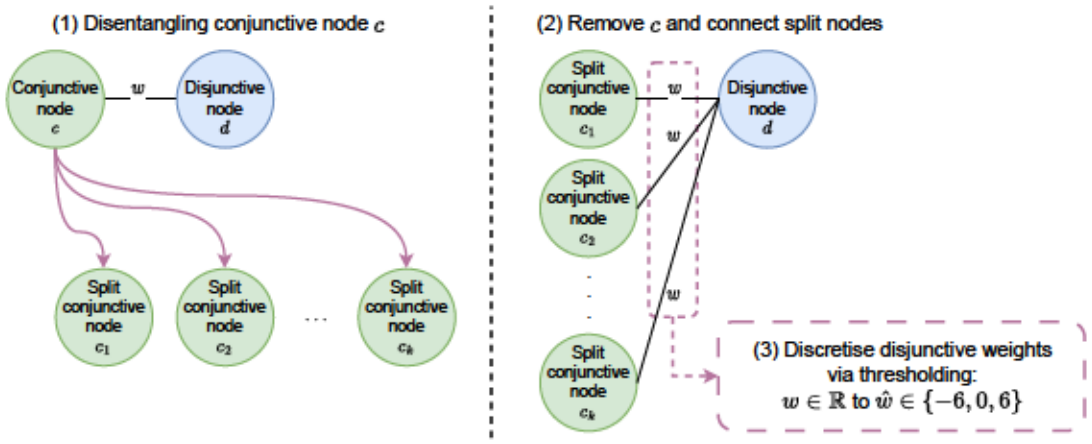


Figure 7.2: The three steps of the disentanglement method. Here w is the weight connecting conjunctive node c to disjunctive node d.

c_ℓ that jointly encode the interpreted behaviour of c, such that for each c_ℓ its associated weights, $\tilde{\mathbf{w}}_\ell$ are discretised, i.e. $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$. The *split weight set* $\mathcal{W} = \{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \ldots\}$ for c is the collection of weights associated with the split nodes $\{c_1, c_2, \ldots\}$.

Recall that the raw output of c for input $\mathbf{x} \in \{-1, 1\}^s$ can be written as:

$$f_{\mathbf{w}}(\mathbf{x}) = \sum_{j \in \mathcal{J}} w_j x_j + \max_{j \in \mathcal{J}} |w_j| - \sum_{j \in \mathcal{J}} |w_j| \tag{7.7}$$

where the relevant set of indices of non-zero weights $\mathcal{J} = \{j \in \{1..s\} \,|\, w_j \neq 0\}$.

For any input $\mathbf{x}$ for which c gives positive output there must be at least one split node c_ℓ with positive output, while the output of all split nodes for any other input must be less than or equal to zero. More formally:

Definition 7.3.1 (Valid Split Weight Set for Positively Connected Conjunctive Node)**.** *A split weight set for a positively connected conjunctive node c, $\mathcal{W} = \{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \ldots\}$ and $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$, is valid if it jointly encodes the interpreted behaviour of the original conjunctive node. It should satisfy the following constraints:*

$$\forall \mathbf{x}^+ \in \mathcal{X}^+ . \exists \ell . f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^+) > 0 \qquad \textit{and} \qquad \forall \mathbf{x}^- \in \mathcal{X}^- . \forall \ell . f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^-) \leq 0 \tag{7.8}$$

where $\mathcal{X}^+ = \{\mathbf{x} \in \{-1, 1\}^s | f_{\mathbf{w}}(\mathbf{x}) > 0\}$ is the set of positive inputs and $\mathcal{X}^- = \mathcal{X} \setminus \mathcal{X}^+$ is the set of negative inputs. Or equivalently, as follows:

$$\forall \mathbf{x} \in \mathcal{X}^+ . \exists \ell \left[\sum_{j \in \mathcal{J}} \tilde{w}_{\ell,j} x_j + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{\ell,j}| \right) = 6 \right] \tag{7.9}$$

$$\forall \mathbf{x} \in \mathcal{X}^- . \neg \exists \ell \left[\sum_{j \in \mathcal{J}} \tilde{w}_{\ell,j} x_j + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{\ell,j}| \right) > -6 \right] \tag{7.10}$$

The first half of constraint in Equation (7.8) means that for every positive example ($\mathbf{x}^+$) at least

one rule encoded by a split node is activated, and the second half means that for every negative example ($\mathbf{x}^-$) none of the rules encoded by the split nodes are activated. These two parts of the constraint in Equation (7.8) are expanded into Equation (7.9) and Equation (7.10) respectively using the property in Proposition 4.1.1 that the discretised conjunctive node output can only be equal to 6 or less than or equal to -6.

Proposition 7.3.2. *A valid split weight set $\mathcal{W}$ for a positively connected conjunctive node c can be constructed by associating each positive example $\mathbf{x}_i \in \mathcal{X}^+$ to a unique split node c_i with weights $\tilde{\mathbf{w}}_i$, where:*

$$\forall j \in \{1 \dots s\}. \tilde{w}_{i,j} = x_{i,j} \cdot \mathbb{1}\left(x_{i,j} = sign(w_j)\right) \cdot 6 = \begin{cases} 6 \cdot x_{i,j} & \text{if } x_{i,j} = sign(w_j) \\ 0 & \text{otherwise} \end{cases} \tag{7.11}$$

This valid split weight set $\mathcal{W}$ constructed under Proposition 7.3.2 has the cardinality of $|\mathcal{X}^+|$. And $\mathcal{W}$ can be translated to a set of rules $\mathfrak{L}$, in the form of:

$$\mathfrak{L} = \left\{ \mathrm{c} \leftarrow \bigwedge_{j \in \mathcal{J}^+_{\tilde{\mathbf{w}}_\ell}} \mathrm{a}_j, \bigwedge_{j \in \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell}} \texttt{not a}_j \;\middle|\; \ell \in \{1..|\mathcal{X}^+|\} \right\} \tag{7.12}$$

where $\mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \{j \in \mathcal{J} | \tilde{w}_{\ell,j} = 6\}, \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \{j \in \mathcal{J} | \tilde{w}_{\ell,j} = -6\}$.

Proof sketch of Proposition 7.3.2. This proof sketch shows that $\mathcal{W}$ constructed under Proposition 7.3.2 satisfies the constraints in Equation (7.8): the positive coverage is straightforward and the negative coverage is proved by contradiction.

Case 1: Positive Coverage – All Positive Examples are Covered (Constraint in Equation (7.9))

For any positive example $\mathbf{x}_i \in \mathcal{X}^+$, every active index contributes 6 to both the weighted sum and the bias term in Equation (4.1), so $f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_i) = 6$ and Equation (7.9) holds for all positive examples.

Case 2: Negative Coverage – No Negative Examples are Covered (Constraint in Equation (7.10))

Suppose $\mathbf{x}_m \in \mathcal{X}^-$ is covered by $\tilde{\mathbf{w}}_i$ associated with $\mathbf{x}_i \in \mathcal{X}^+$. Then $f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_m) = 6$, which simplifies to $\sum_j \tilde{w}_{i,j} x_{m,j} = \sum_j |\tilde{w}_{i,j}|$. Substituting Equation (7.11) forces $\mathbf{x}_m$ to agree with $\mathbf{x}_i$ on every index where $x_{i,j} = \text{sign}(w_j)$. Because $\mathbf{x}_i \neq \mathbf{x}_m$, there exists an index with $x_{m,j} = \text{sign}(w_j)$ and $x_{i,j} \neq \text{sign}(w_j)$. Decomposing $f_{\mathbf{w}}(\mathbf{x}_m)$ and $f_{\mathbf{w}}(\mathbf{x}_i)$ by sign agreement with $\mathbf{w}$ then yields $f_{\mathbf{w}}(\mathbf{x}_m) > f_{\mathbf{w}}(\mathbf{x}_i) > 0$, contradicting $\mathbf{x}_m \in \mathcal{X}^-$. Hence no negative example is covered.

A complete proof, including the set-decomposition and inequality steps, is given in Appendix D. ■

7.3.2 Disentangling a Negatively Connected Conjunctive Node

This subsection mirrors the structure of Section 7.3.1, providing intuition for Proposition 7.3.3 followed by a proof sketch. The complete proof is in Appendix D.

When the conjunctive node is connected to the disjunctive node with a negative weight, the disjunctive node will fire only when the conjunction is not true. Given a single conjunctive node, c, parameterised by $\mathbf{w} \in \mathbb{R}^s$ with a total of s inputs and connected to a disjunctive node with a *negative* weight, the goal is to find a collection of discretised split nodes c_ℓ that jointly encode the *negated* interpreted behaviour of c. That is, find any input $\mathbf{x}$ for which c gives non-positive output

(the conjunction is not satisfied) and ensure that at least one split node c_ℓ propagates a positive signal to the disjunctive node so that this disjunction can fire, while the output of all split nodes for any positive input must be less than or equal to zero. Again, the *split weight set* $\mathcal{W} = \{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \ldots\}$ for c is the collection of discretised weights associated with the split nodes $\{c_1, c_2, \ldots\}$, i.e. for each c_ℓ its associated weights $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$.

Definition 7.3.2 (Valid Split Weight Set for Negatively Connected Conjunctive Node)**.** *A split weight set for a negatively connected conjunctive node c, $\mathcal{W} = \{\tilde{\mathbf{w}}_1, \tilde{\mathbf{w}}_2, \ldots\}$ and $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$, is valid if it jointly encodes the negated interpreted behaviour of the original conjunctive node. It should satisfy the following constraints:*

$$\forall \mathbf{x}^- \in \mathcal{X}^-. \exists \ell. f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^-) > 0 \qquad \textit{and} \qquad \forall \mathbf{x}^+ \in \mathcal{X}^+. \forall \ell. f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^+) \leq 0 \tag{7.13}$$

where $\mathcal{X}^+ = \{\mathbf{x} \in \{-1, 1\}^s | f_{\mathbf{w}}(\mathbf{x}) > 0\}$ is the set of positive inputs and $\mathcal{X}^- = \mathcal{X} \setminus \mathcal{X}^+$ is the set of negative inputs. Or equivalently, as follows:

$$\forall \mathbf{x} \in \mathcal{X}^-. \exists \ell \left[\sum_{\jmath \in \mathcal{J}} \tilde{w}_{\ell,\jmath} x_\jmath + \left(6 - \sum_{\jmath \in \mathcal{J}} |\tilde{w}_{\ell,\jmath}| \right) = 6 \right] \tag{7.14}$$

$$\forall \mathbf{x} \in \mathcal{X}^+. \neg \exists \ell \left[\sum_{\jmath \in \mathcal{J}} \tilde{w}_{\ell,\jmath} x_\jmath + \left(6 - \sum_{\jmath \in \mathcal{J}} |\tilde{w}_{\ell,\jmath}| \right) > -6 \right] \tag{7.15}$$

The two parts of the constraint in Equation (7.13) are expanded into Equation (7.14) and Equation (7.15) respectively using the property in Proposition 4.1.1 that the discretised conjunctive node output can only equal 6 or be less than or equal to -6.

Note that the conditions are reversed compared to the positive case in Section 7.3.1. If an input results in a less-or-equal-to-0 output for conjunctive node with $\mathbf{w}$, the connected disjunctive node will fire, and a split node that fires under this input is needed to connect it to the same disjunctive node but positively (Equation (7.14)). Similarly, no split nodes should cover any of the positive examples so that the disjunctive node will not fire (Equation (7.15)).

Proposition 7.3.3. *A valid split weight set $\mathcal{W}$ for a negatively connected conjunctive node c can be constructed by associating each negative example $\mathbf{x}_\imath \in \mathcal{X}^-$ to a unique split node $c_\imath$ with weights $\tilde{\mathbf{w}}_\imath$, where:*

$$\forall j \in \{1 \ldots s\}. \tilde{w}_{\imath,\jmath} = x_{\imath,\jmath} \cdot \mathbb{I}\left(x_{\imath,\jmath} \neq sign(w_\jmath)\right) \cdot 6 = \begin{cases} 6 \cdot x_{\imath,\jmath} & \text{if } x_{\imath,\jmath} \neq sign(w_\jmath) \\ 0 & \text{otherwise} \end{cases} \tag{7.16}$$

This valid split weight set $\mathcal{W}$ constructed under Proposition 7.3.3 has the cardinality of $|\mathcal{X}^-|$. And $\mathcal{W}$ can be translated to a set of rules $\mathfrak{L}$, in the form of:

$$\mathfrak{L} = \left\{ \mathrm{c} \leftarrow \bigwedge_{\jmath \in \mathcal{J}^+_{\tilde{\mathbf{w}}_\ell}} \mathrm{a}_\jmath, \bigwedge_{\jmath \in \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell}} \texttt{not } \mathrm{a}_\jmath \;\middle|\; \ell \in \{1..|\mathcal{X}^-|\} \right\} \tag{7.17}$$

where $\mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \{j \in \mathcal{J} | \tilde{w}_{\ell,\jmath} = 6\}, \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \{j \in \mathcal{J} | \tilde{w}_{\ell,\jmath} = -6\}$.

Proof sketch of Proposition 7.3.3. This proof shows that $\mathcal{W}$ constructed under Proposition 7.3.3 satisfies the constraints in Equation (7.13): the negative coverage is straightforward and the positive coverage is proved by contradiction.

Case 1: Negative Coverage – All Negative Examples are Covered (Constraint in Equation (7.14))

For any negative example $\mathbf{x}_i \in \mathcal{X}^-$, every active index contributes 6 to both the weighted sum and the bias term in Equation (4.1), so $f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_i) = 6$ and Equation (7.14) holds for all negative examples.

Case 2: Positive Coverage – No Positive Examples are Covered (Constraint in Equation (7.15))

Suppose $\mathbf{x}_m \in \mathcal{X}^+$ is covered by $\tilde{\mathbf{w}}_i$ associated with $\mathbf{x}_i \in \mathcal{X}^-$. The same coverage identity $\sum_j \tilde{w}_{i,j} x_{m,j} = \sum_j |\tilde{w}_{i,j}|$ applies; with Equation (7.16) this forces $\mathbf{x}_m$ to agree with $\mathbf{x}_i$ on every index where $x_{i,j} \neq \text{sign}(w_j)$. Because $\mathbf{x}_i \neq \mathbf{x}_m$, there exists an index with $x_{m,j} \neq \text{sign}(w_j)$ and $x_{i,j} = \text{sign}(w_j)$. Sign-based decomposition of $f_{\mathbf{w}}(\mathbf{x}_m)$ and $f_{\mathbf{w}}(\mathbf{x}_i)$ then yields $f_{\mathbf{w}}(\mathbf{x}_m) < f_{\mathbf{w}}(\mathbf{x}_i) \leq 0$, contradicting $\mathbf{x}_m \in \mathcal{X}^+$. Hence no positive example is covered.

A complete proof, including the set-decomposition and inequality steps, is given in Appendix D. ■

7.3.3 Optimisation

To reduce the size of the split weight set, redundant weight vectors can be identified and removed. A weight vector in the split weight set is redundant if its corresponding rule is logically subsumed by the rule of another weight in the same split weight set. A subsumption relation between two weight vectors in the split weight set can be defined as:

Definition 7.3.3. *A conjunctive semi-symbolic node's split weight $\tilde{\mathbf{w}}_m$ **subsumes** another split weight $\tilde{\mathbf{w}}_n$ if all non-zero weights in $\tilde{\mathbf{w}}_m$ match those corresponding weights in $\tilde{\mathbf{w}}_n$ and there are fewer non-zero weights in $\tilde{\mathbf{w}}_m$ than in $\tilde{\mathbf{w}}_n$:*

$$(\forall j \in \mathcal{J}.\tilde{w}_{m,j} \in \{0, \tilde{w}_{n,j}\}) \wedge (0 < |\{j \in \mathcal{J} : \tilde{w}_{m,j} \neq 0\}| < |\{j \in \mathcal{J} : \tilde{w}_{n,j} \neq 0\}|)$$

Proposition 7.3.4 allows the reduction of $\mathcal{W}$ to a smaller yet equally valid split weight set $\mathcal{W}^*$ where no pair of weight vectors is subsumed by the other.

Proposition 7.3.4. *If a split weight $\tilde{\mathbf{w}}_n$, associated to $\mathbf{x}_n \in \mathcal{X}^+$ when the original conjunctive node is positively used or $\mathbf{x}_n \in \mathcal{X}^-$ when the original conjunctive node is negatively used, is subsumed by another split weight $\tilde{\mathbf{w}}_m$, then $\tilde{\mathbf{w}}_m$ must also cover $\mathbf{x}_n$. That is, $f_{\tilde{\mathbf{w}}_m}(\mathbf{x}_n) = 6$.*

Proof of Proposition 7.3.4. When the original conjunctive node is positively used and $\tilde{\mathbf{w}}_n$ is associ-

ated to $\mathbf{x}_n \in \mathcal{X}^+$, the following holds:

$$\begin{aligned}
f_{\tilde{\mathbf{w}}_m}(\mathbf{x}_n) &= \sum_{j \in \mathcal{J}} \tilde{w}_{m,j} x_{n,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{m,j}|\right) \\
&= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \tilde{w}_{n,j} x_{n,j} \\
&\quad + \left(6 - \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) |\tilde{w}_{n,j}|\right) && \text{by Def. (7.3.3)} \\
&= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \mathbb{1}\left(x_{n,j} = \operatorname{sign}(w_j)\right) \\
&\quad + \left(6 - 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \mathbb{1}\left(x_{n,j} = \operatorname{sign}(w_j)\right)\right) && \text{by Eq. (7.11)} \\
&= 6
\end{aligned}$$

Thus, $\tilde{\mathbf{w}}_m$ covers $\mathbf{x}_n$.

When the original conjunctive node is negatively used and $\tilde{\mathbf{w}}_n$ is associated to $\mathbf{x}_n \in \mathcal{X}^-$, the following holds:

$$\begin{aligned}
f_{\tilde{\mathbf{w}}_m}(\mathbf{x}_n) &= \sum_{j \in \mathcal{J}} \tilde{w}_{m,j} x_{n,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{m,j}|\right) \\
&= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \tilde{w}_{n,j} x_{n,j} \\
&\quad + \left(6 - \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) |\tilde{w}_{n,j}|\right) && \text{by Def. (7.3.3)} \\
&= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \mathbb{1}\left(x_{n,j} \neq \operatorname{sign}(w_j)\right) + \\
&\quad + \left(6 - 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(\tilde{w}_{m,j} = \tilde{w}_{n,j}\right) \mathbb{1}\left(x_{n,j} \neq \operatorname{sign}(w_j)\right)\right) && \text{by Eq. (7.16)} \\
&= 6
\end{aligned}$$

Thus, $\tilde{\mathbf{w}}_m$ covers $\mathbf{x}_n$. ■

While some weights are not essential for example coverage, the naive computation of the minimal split weight set $\mathcal{W}^*$ by computing all possible split weights and selecting the most general ones is not efficient. To optimise the computation, the properties of the conjunctive node's output are first analysed.

Remark 7.3.1. *Given a sample* $\mathbf{x}^{(k)} \in \{-1, 1\}^s$*, recall the sign-matching and sign-mismatching*

sets defined in Equation (4.9) and Equation (4.10) respectively:

$$\mathcal{J}^{+}_{\mathbf{w}\odot\mathbf{x}^{(k)}} = \left\{ j \in \mathcal{J} \;\middle|\; w_j x_j^{(k)} > 0 \right\}$$
$$\mathcal{J}^{-}_{\mathbf{w}\odot\mathbf{x}^{(k)}} = \left\{ j \in \mathcal{J} \;\middle|\; w_j x_j^{(k)} < 0 \right\}$$
$$\mathcal{J}^{+}_{\mathbf{w}\odot\mathbf{x}^{(k)}} \cup \mathcal{J}^{-}_{\mathbf{w}\odot\mathbf{x}^{(k)}} = \mathcal{J}$$

Equation (7.7) can be rewritten as:

$$f_{\mathbf{w}}\left(\mathbf{x}^{(k)}\right) = \max_{j\in\mathcal{J}} |w_j| - 2 \sum_{j\in\mathcal{J}^{-}_{\mathbf{w}\odot\mathbf{x}^{(k)}}} |w_j|$$

Given $\mathbf{x}^{(k)}$, the bivalent interpretation of this node $b^{(k)} = \top$ when $f_{\mathbf{w}}(\mathbf{x}^{(k)}) > 0$. In the context of splitting a positively used conjunctive node, the following property is relevant:

$$f_{\mathbf{w}}\left(\mathbf{x}^{(k)}\right) > 0 \iff \max_{j\in\mathcal{J}} |w_j| > 2 \sum_{j\in\mathcal{J}^{-}_{\mathbf{w}\odot\mathbf{x}^{(k)}}} |w_j| \tag{7.18}$$

The bivalent interpretation of this node $b^{(k)} = \bot$ when $f_{\mathbf{w}}(\mathbf{x}^{(k)}) \leq 0$. In the context of splitting a negatively used conjunctive node, the following property is relevant:

$$f_{\mathbf{w}}\left(\mathbf{x}^{(k)}\right) \leq 0 \iff \max_{j\in\mathcal{J}} |w_j| \leq 2 \sum_{j\in\mathcal{J}^{-}_{\mathbf{w}\odot\mathbf{x}^{(k)}}} |w_j| \tag{7.19}$$

Optimisation on Splitting Positively-used Conjunctive Node

Consider the case where the conjunctive node is positively used. Recall the translated rule set $\mathfrak{L}$ that covers the soft-valued truth table of the original conjunctive node, indexed by ℓ, defined in Equation (7.12). And recall the split sets of the relevant set $\mathcal{J}$ in Definition 4.1.4, $\mathcal{J}^{+} = \{j \in \mathcal{J} | w_j > 0\}$ and $\mathcal{J}^{-} = \{j \in \mathcal{J} | w_j < 0\}$ that separate the positive and negative weights. Following the splitting method described in Section 7.3.1, at least one of the rules in $\mathfrak{L}$ will be triggered for a positive example $\mathbf{x}^{(n)}$, i.e. there exists ℓ such that $f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^{(n)}) > 0$ and Rule ℓ is triggered by $\mathbf{x}^{(n)}$.

The intuition is that a weight with a 'large' absolute value indicates a heavy contribution from the input to the output bivalent interpretation, and if the input and this weight have opposite signs, the contribution is negative and makes the output bivalent interpretation false. Therefore, every 'large' positive weight should be present in all rules' positive sets of body atoms, i.e. $\mathcal{J}^{+}_{\tilde{\mathbf{w}}_\ell}$ from Equation (7.12) for any ℓ. Mathematically, using Remark 7.3.1, for any $j \in \mathcal{J}$ such that $w_j \geq 1/2 \max_{j\in\mathcal{J}} |w_j|$ (being 'large' and positive), j should not be in any sign-mismatching set $\mathcal{J}^{-}_{\tilde{\mathbf{w}}_\ell\odot\mathbf{x}^{(n)}}$ for any positive example $\mathbf{x}^{(n)}$ and ℓ, so that the property in Equation (7.18) holds. Thus, j must be in the sign-matching set $\mathcal{J}^{+}_{\tilde{\mathbf{w}}_\ell\odot\mathbf{x}^{(n)}}$ for any ℓ and $\mathbf{x}^{(n)}$. Because the split weight set $\mathcal{W}$ is constructed by associating each positive example $\mathbf{x}^{(n)}$ to a split node with weights $\tilde{\mathbf{w}}_\ell$ under the sign-matching rule in Equation (7.11), $\tilde{w}_{\ell,j}$ will be positive and thus the j-th atom will be present and positive in all rules' positive sets of body atoms. That is, $\forall j \forall \ell. \left[w_j \geq 1/2 \max_{j\in\mathcal{J}} |w_j| \implies j \in \mathcal{J}^{+}_{\tilde{\mathbf{w}}_\ell} \right]$. Similarly, for a 'large' negative weight $w_j \leq -1/2 \max_{j\in\mathcal{J}} |w_j|$, $j \in \mathcal{J}^{-}_{\tilde{\mathbf{w}}_\ell}$ for all ℓ. This property can be formalised as:

Remark 7.3.2. *The following sets of indices are defined based on the absolute values of the weights:*

$$\bar{\mathcal{J}} = \left\{ j \in \mathcal{J} \;\middle|\; |w_j| < \frac{1}{2} \max_{j' \in \mathcal{J}} |w_{j'}| \right\}$$

$\bar{\mathcal{J}}^+ = \bar{\mathcal{J}} \cap \mathcal{J}^+$ *small but positive weights*

$\bar{\mathcal{J}}^- = \bar{\mathcal{J}} \cap \mathcal{J}^-$ *small but negative weights*

If the conjunctive node is positively used, the indices where the weights' absolute values are greater than half of the maximum absolute weight should be included in all split weight vectors' relevant sets and thus present in all split rules' body atoms:

$$\forall \ell . j . \left[j \in (\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+) \implies j \in \mathcal{I}^+_{\tilde{\mathbf{w}}_\ell} \right]$$
$$\forall \ell . j . \left[j \in (\mathcal{J}^- \setminus \bar{\mathcal{J}}^-) \implies j \in \mathcal{I}^-_{\tilde{\mathbf{w}}_\ell} \right]$$

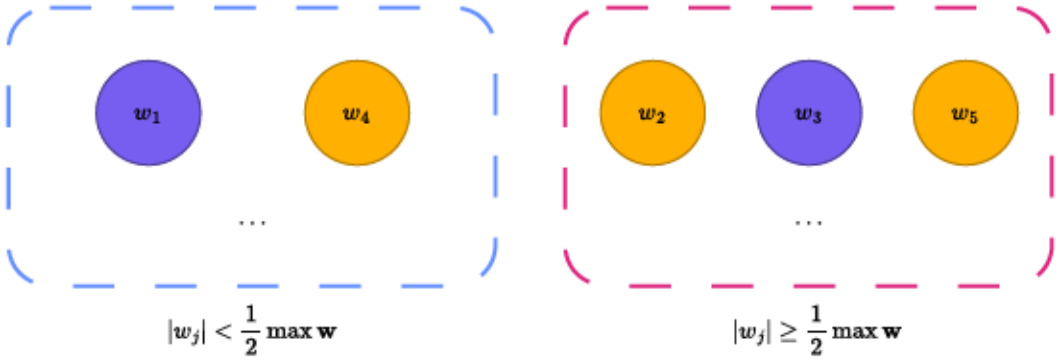


Figure 7.3: Split elements in the original conjunctive node's weights $\mathbf{w}$ into two sets, based on the absolute values of the elements. The original sign of the weights is preserved in the figure, where the purple weights are positive and the yellow weights are negative. The set on the left is the small-magnitude weights where $|w_j| < \max |\mathbf{w}|/2$, and the set on the right is the large-magnitude weights where $|w_j| \geq \max |\mathbf{w}|/2$. $\bar{\mathcal{J}}^+$ is the set of indices correspond to the purple weights in the set on the left, and $\bar{\mathcal{J}}^-$ is the set of indices correspond to the yellow weights in the set on the left. $\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+$ is the set of indices of the weights in the set on the right that are positive (purple), and $\mathcal{J}^- \setminus \bar{\mathcal{J}}^-$ is the set of indices of the weights in the set on the right that are negative (yellow). All the weights in the set on the right should be present in all the split weight vectors in the split weight set $\mathcal{W}$.

Using the properties above, a three-step optimisation process is introduced to compute the *optimal split weight set* $\mathcal{W}^*$ for a positively used conjunctive node:

1. **Compute the set of candidate indices of all small-magnitude weights in the original node that might not be essential to cover any positive examples.** This step is the execution of Remark 7.3.1 and 7.3.2. Weights with large absolute values contribute more to the output bivalent interpretation, and thus every large positive weight should be in all rules' positive sets of atoms $\mathcal{I}^+_{\tilde{\mathbf{w}}_\ell}$ and every negative weight with a large absolute value should be in all rules' negative sets of atoms $\mathcal{I}^-_{\tilde{\mathbf{w}}_\ell}$. A weight w_j is considered as having a large magnitude if $|w_j| \geq \max |\mathbf{w}|/2$, meaning $\bar{\mathcal{J}} = \{j \in \mathcal{J} \mid |w_j| < \max |\mathbf{w}|/2\}$ contains all the small-magnitude weights that might not be essential to cover any positive examples.

2. **Identify optimal combinations of candidate indices that can be excluded.** An exclusion set $\mathcal{E}_\ell$ of a weight vector $\tilde{\mathbf{w}}_\ell$ is defined as the set of indices of the non-zero elements in the original $\mathbf{w}$ which were excluded by the disentanglement process when constructing $\tilde{\mathbf{w}}_\ell$, that is, $\{j \in \mathcal{J} | \tilde{w}_{\ell,j} = 0 \wedge w_j \neq 0\}$. By Remark 7.3.1, the definition can be reformulated: a set $\mathcal{E}$ is an exclusion set if and only if $\sum_{j\in\mathcal{E}} |w_j| < \max|\mathbf{w}|/2$. Since $\max|\mathbf{w}|/2$ is also the threshold value used to define $\bar{\mathcal{J}}$, an exclusion set $\mathcal{E}$ must be a subset of $\bar{\mathcal{J}}$, although a subset of $\mathcal{J}$ is not necessarily an exclusion set. If $\tilde{\mathbf{w}}_\ell$ subsumes $\tilde{\mathbf{w}}_{\ell'}$, every zero element in $\tilde{\mathbf{w}}_{\ell'}$ must also be zero in $\tilde{\mathbf{w}}_\ell$, and $\tilde{\mathbf{w}}_\ell$ must have more zero elements than $\tilde{\mathbf{w}}_{\ell'}$, thus $\mathcal{E}_\ell \supset \mathcal{E}_{\ell'}$. The original optimisation problem over finding $\mathcal{W}^*$ can therefore be transformed to finding $E^* = \{\mathcal{E}_1^*, \ldots, \mathcal{E}_\ell^*, \ldots, \mathcal{E}_L^*\}$ where no pairs of elements are subsets of each other. To find E^*, a search can be performed by considering possible candidate $\hat{\mathcal{J}} \subset \mathcal{J}$ and verifying it is an exclusion set using $\sum_{j\in\hat{\mathcal{J}}} |w_j| < \max|\mathbf{w}|/2$. Full details of this search are in Algorithm 5.

3. **Convert exclusion sets to weight vectors.** For each exclusion set $\mathcal{E}_\ell^* \in E^*$, a new weight vector $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$ is created where its j-th element is:

$$\tilde{w}_{\ell,j} = \begin{cases} 6 \cdot \text{sign}(w_j) & \text{if } j \in \mathcal{J} \setminus \mathcal{E}_\ell \\ 0 & \text{otherwise} \end{cases}$$

Thus, $\mathcal{W}^* = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_L\}, L \leq |\mathcal{X}^+|$ is computed for a positively used conjunctive node, where no pair of weight vectors is subsumed by the other. Below is an example of how the optimisation process works:

Example 7.3.1: Finding split weight set for a positively used conjunctive node

Recall the conjunctive node in Example 4.1.1 with weights $\mathbf{w} = [-6, -2, -2, 2, -6]$, and assume that the original node is positively connected to a disjunctive node. The small-magnitude weight set $\bar{\mathcal{J}}$ is computed as (using one-indexing):

$$\frac{1}{2}\max|\mathbf{w}| = 3$$
$$\bar{\mathcal{J}} = \{j \in \mathcal{J} \mid |w_j| < \frac{1}{2}\max|\mathbf{w}|\} = \{2, 3, 4\}$$

And $\{1, 5\}$ are the indices of the large-magnitude weights that should be included in all rules' body atoms. In particular, for all ℓ, $\{1, 5\} \subseteq \mathcal{J}_{\tilde{\mathbf{w}}_\ell}^-$.
There are three optimal exclusion sets, where each exclusion set is a strict subset of $\mathcal{J}$:

$$\mathcal{E}_1^* = \{2\} \qquad \sum_{j\in\mathcal{E}_1^*} |w_j| = 2 < 3$$
$$\mathcal{E}_2^* = \{3\} \qquad \sum_{j\in\mathcal{E}_2^*} |w_j| = 2 < 3$$
$$\mathcal{E}_3^* = \{4\} \qquad \sum_{j\in\mathcal{E}_3^*} |w_j| = 2 < 3$$

The corresponding split weight vectors are computed as:

$$\begin{aligned} \mathcal{J} \setminus \mathcal{E}_1^* &= \{1,3,4,5\} & \tilde{\mathbf{w}}_1 &= [-6,0,-6,6,-6] \\ \mathcal{J} \setminus \mathcal{E}_2^* &= \{1,2,4,5\} & \tilde{\mathbf{w}}_2 &= [-6,-6,0,6,-6] \\ \mathcal{J} \setminus \mathcal{E}_3^* &= \{1,2,3,5\} & \tilde{\mathbf{w}}_3 &= [-6,-6,-6,0,-6] \end{aligned}$$

And the translated rules are:

$$\mathfrak{L}_{\mathcal{W}^\bullet} = \left\{ \begin{array}{l} \mathtt{c} \leftarrow \mathtt{not\ a_1, not\ a_3, a_4, not\ a_5.} \\ \mathtt{c} \leftarrow \mathtt{not\ a_1, not\ a_2, a_4, not\ a_5.} \\ \mathtt{c} \leftarrow \mathtt{not\ a_1, not\ a_2, not\ a_3, not\ a_5.} \end{array} \right\}.$$

This rule set matches the rule set $\mathfrak{L}$ in Equation (7.1) (in a different order of the rules but the same rules). Both rule sets perfectly describe the node's behaviour.

Algorithm 5 The full algorithm on splitting a positively used conjunctive node with optimisation

Input: Weight $\mathbf{w}$ of a conjunctive node positively connected to a disjunctive node
Output: List of valid split weights

Compute $\mathcal{J} \leftarrow \{j \in \{1..s\} | w_j \neq 0\}$
Compute $\bar{\mathcal{J}} \leftarrow \{j \in \mathcal{J} \mid |w_j| < \max|\mathbf{w}|/2\}$ ▷ See Remark 7.3.2
if $\mathcal{J} = \emptyset$ **then**
    return $[\text{sign}(\mathbf{w}) \cdot 6]$
/* Use breadth-first search to explore subsets of $\bar{\mathcal{J}}$ that are valid exclusion candidates $\mathcal{E}$ while maximising its cardinality, where: $\sum_{j \in \mathcal{E}} |w_j| < \max|\mathbf{w}|/2$ */
valid_splits $\leftarrow \{\}$, *queue* $\leftarrow$ empty OrderedDict
/* Add all $j \in \bar{\mathcal{J}}$ as a set $\{j\}$ to *queue* with parent = None */
for $j \in \bar{\mathcal{J}}$ **do**
    queue.add($\{j\}$, parent = None)
while *queue* is not empty **do**
    Get the head item (*removal_indices*, *parent*) from *queue*
    new_half_max $\leftarrow \max|\mathbf{w}|/2 - \sum|\mathbf{w}[\textit{removal_indices}]|$
    $\mathcal{J}_{new} \leftarrow \{j \in \mathcal{J} \mid j \notin$ *removal_indices* and $|w_j| <$ *new_half_max*$\}$
    if *new_half_max* ≤ 0 **or** $\mathcal{J}_{new} = \emptyset$ **then**
        /* The current *removal_indices* is not a valid exclusion set or there are no more elements in $\hat{\mathcal{J}}$ that can be added to the exclusion set */
        if *new_half_max* > 0 **then**
            valid_splits.add(*removal_indices*)
        else if *parent* is not None **then**
            valid_splits.add(*parent*)
        continue
    /* Keep exploring bigger exclusion sets */
    $\mathcal{J}_{new} \leftarrow \mathcal{J}_{new} -$ *removal_indices*
    if $\mathcal{J}_{new}$ is empty **then**
        valid_splits.add(*removal_indices*)
        continue
    for $j \in \mathcal{J}_{new}$ **do**
        new_removal $\leftarrow$ sort(*removal_indices* $\cup \{j\}$)
        if *new_removal* $\notin$ *queue* **then**
            queue.add(*new_removal*, parent = *removal_indices*)
split_tensors $\leftarrow []$
for each valid exclusion set $\mathcal{E}$ **do**
    Create new tensor $\tilde{\mathbf{w}}$ where:
    - $j \in \mathcal{E}$: $\tilde{w}_j = 0$
    - $j \in \mathcal{J} \setminus \mathcal{E}$: $\tilde{w}_j = 6 \cdot \text{sign}(w_j)$
    Add $\tilde{\mathbf{w}}$ to *split_tensors*
return *split_tensors*

Optimisation on Splitting Negatively-used Conjunctive Node

Consider the case where the conjunctive node is negatively used. Following the splitting method described in Section 7.3.2, at least one of the rules in $\mathfrak{L}$ will be triggered for any negative example $\mathbf{x}^{(m)} \in \mathcal{X}^-$, i.e. there exists ℓ such that $f_{\tilde{\mathbf{w}}_\ell}(\mathbf{x}^{(m)}) > 0$ even though the original node produces $f_{\mathbf{w}}(\mathbf{x}^{(m)}) \leq 0$, and Rule ℓ is triggered by $\mathbf{x}^{(m)}$.

The intuition is that a weight with a 'large' absolute value indicates a heavy contribution from the input to the output bivalent interpretation, and if the input and this weight have opposite signs, this single contribution alone is sufficient to make the output bivalent interpretation false. Therefore, any 'large' weight can be the only element to appear in the split weight vector's relevant set but negated, and the split rule's body contains only the single negated atom corresponding to that weight. Mathematically, using Remark 7.3.1, for any $j \in \mathcal{J}$ such that $|w_j| \geq 1/2 \max_{\jmath \in \mathcal{J}} |w_\jmath|$ (being 'large'), if the sign of $x_j^{(m)}$ does not match the sign of w_j, then $x_j^{(m)} \cdot w_j$ alone exceeds the threshold in Equation (7.19), causing the original conjunctive node's output to be non-positive. Consequently, for such a large positive weight $w_j \geq 1/2 \max_{\jmath \in \mathcal{J}} |w_\jmath|$, there exists an ℓ where the j-th atom appears as a single negated atom, i.e. $\mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \{j\}$ and $\mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \emptyset$. Similarly, for a 'large' negative weight $w_j \leq -1/2 \max_{\jmath \in \mathcal{J}} |w_\jmath|$, there exists an ℓ where $\mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \{j\}$ and $\mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \emptyset$. This property can be formalised as:

Remark 7.3.3. *If the conjunctive node is negatively used, the indices where the weights' absolute values are greater than or equal to half of the maximum absolute weight can each form a singleton rule body, with the atom appearing in its negated form (opposite to the weight's sign):*

$$\forall j. \left[j \in (\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+) \implies \exists \ell. \, \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \{j\} \wedge \mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \emptyset \right]$$
$$\forall j. \left[j \in (\mathcal{J}^- \setminus \bar{\mathcal{J}}^-) \implies \exists \ell. \, \mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} = \{j\} \wedge \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell} = \emptyset \right]$$

where $\bar{\mathcal{J}}^+$ and $\bar{\mathcal{J}}^-$ are the same sets as defined in Remark 7.3.2.

Using the properties above, a three-step optimisation process is introduced to compute the *optimal split weight set* $\mathcal{W}^*$ for a negatively used conjunctive node:

1. **Compute the singleton sets and the set of candidate indices of all small-magnitude weights.** Each weight with a large absolute values ($|w_j| > \max|\mathbf{w}|/2$)[1] is enough to make the original conjunctive node's behaviour false if its sign does not match with its input, and thus it can be the only non-zero element in a split weight vector. An inclusion set $\mathcal{O}_\ell$ of a split weight vector $\tilde{\mathbf{w}}_\ell$ is defined as the set of indices of the non-zero elements in the original weight vector $\mathbf{w}$ which were included by the disentanglement process when constructing $\tilde{\mathbf{w}}_\ell$, that is, $\{j \in \mathcal{J} | \tilde{w}_{\ell,j} \neq 0 \wedge w_j \neq 0\}$. By Remark 7.3.3, each weight that satisfies $|w_j| > \max|\mathbf{w}|/2$ will be the single element in an inclusion set $\mathcal{O}_\ell$. The rest of the weights, which might not be essential to cover any negative example, belongs to the small-magnitude weight set $\mathcal{J} = \{j \in \mathcal{J} \mid |w_j| \leq \max|\mathbf{w}|/2\}$.

2. **Identify optimal combinations of candidate indices that can be included.** The minimal set of weights that together can make the original conjunctive node's behaviour false

[1] While $|w_j| = \max|\mathbf{w}|/2$ is enough to make the original conjunctive node's output to be 0 and thus interpreted as false under Remark 7.3.1, in practice the output of the conjunctive node should be non-zero so that the disjunctive layer's input values can be as close to the desirable discrete set of ± 1 as possible.

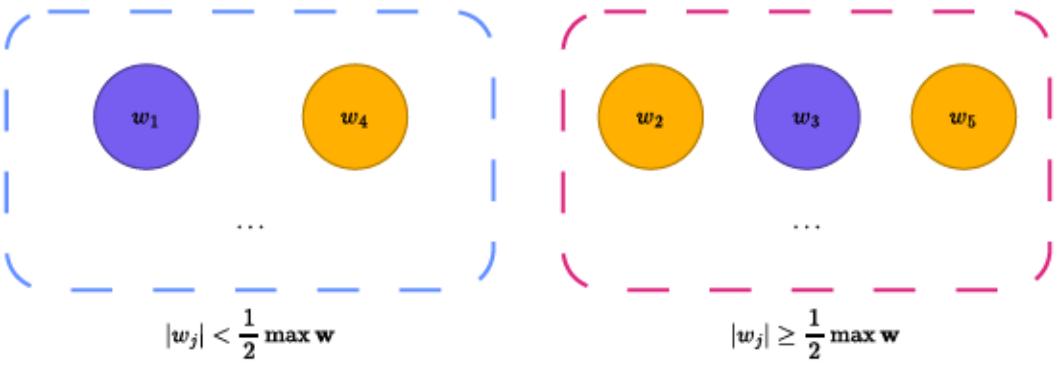


Figure 7.4: Split elements in the original conjunctive node's weights $\mathbf{w}$ into two sets, based on the absolute values of the elements. The original sign of the weights is preserved in the figure, where the purple weights are positive and the yellow weights are negative. The set on the left is the small-magnitude weights where $|w_j| < \max|\mathbf{w}|/2$, and the set on the right is the large-magnitude weights where $|w_j| \geq \max|\mathbf{w}|/2$. $\mathcal{J}^+$ is the set of indices correspond to the purple weights in the set on the left, and $\mathcal{J}^-$ is the set of indices correspond to the yellow weights in the set on the left. $\mathcal{J}^+ \setminus \bar{\mathcal{J}}^+$ is the set of indices of the weights in the set on the right that are positive (purple), and $\mathcal{J}^- \setminus \bar{\mathcal{J}}^-$ is the set of indices of the weights in the set on the right that are negative (yellow). All the weights in the set on the right alone are sufficient to make the original conjunctive node's behaviour false. Each weight in the set on the right can be the only element present in a split weight vector in the split weight set $\mathcal{W}$.

(validity) is found by expanding a candidate inclusion set element-by-element from the small-magnitude weight set $\mathcal{J}$ (optimality).

- Validity: an inclusion set $\mathcal{O}$ is valid if its absolute weights' sum is greater than $\max|\mathbf{w}|/2$, that is, when all the weights in this inclusion set have their signs mismatching the signs of their corresponding input, the output of the original conjunctive node will be negative. Only valid inclusion sets that cover negative examples will be converted to split weights.
- Optimality: by definition, a valid inclusion set contains all the indices of the weights that will be included in the split weight $\tilde{\mathbf{w}}_\ell$, and $\mathcal{O}_\ell = \mathcal{J}^+_{\tilde{\mathbf{w}}_\ell} \cup \mathcal{J}^-_{\tilde{\mathbf{w}}_\ell}$. The fewer elements in the inclusion set, the more 'general' its corresponding rule is and its corresponding split weight subsumes others.

Full details of this expansion process are in Algorithm 6.

3. **Convert inclusion sets to weight vectors.** For each valid and optimal inclusion set $\mathcal{O}^*_\ell \in O^*$, a new weight vector $\tilde{\mathbf{w}}_\ell \in \{-6, 0, 6\}^s$ is created where its j-th element is:

$$\tilde{w}_{\ell,j} = \begin{cases} -6 \cdot \text{sign}(w_j) & \text{if } j \in \mathcal{O}^*_\ell \\ 0 & \text{otherwise} \end{cases}$$

Thus, $\mathcal{W}^* = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_L\}, L \leq |\mathcal{X}^-|$ is computed for a negatively used conjunctive node, where no pair of weight vectors is subsumed by the other. Below is an example of how the optimisation process works:

Example 7.3.2: Finding split weight set for a negatively used conjunctive node

Consider a conjunctive node with weights $\mathbf{w} = [-1.3, -2.2, 6]$ that is negatively connected to a disjunctive node. The bias is calculated as $\beta = \max|\mathbf{w}| - \sum|\mathbf{w}| = 6 - (1.3 + 2.2 + 6) = -3.5$. The soft-valued truth table of this node is:

x_1	x_2	x_3	Node activation $\tanh(\sum_{i=1}^{3} w_i x_i + \beta)$	Bivalent Interpretation b
1	1	1	-0.762	$\bot$
1	1	-1	-1.000	$\bot$
1	-1	1	0.998	$\top$
1	-1	-1	-1.000	$\bot$
-1	1	1	0.922	$\top$
-1	1	-1	-1.000	$\bot$
-1	-1	1	1.000	$\top$
-1	-1	-1	-1.000	$\bot$

The minimal set of rules that perfectly describes the node's negated behaviour (the bivalent interpretation is $\bot$ and marked in blue), which will positively activate its connected disjunctive node, is:

$$\mathfrak{L} = \left\{ \begin{array}{l} \mathrm{c} \leftarrow \mathtt{not}\ \mathrm{a}_3. \\ \mathrm{c} \leftarrow \mathrm{a}_1, \mathrm{a}_2. \end{array} \right\}. \tag{7.20}$$

The small-magnitude weight set $\bar{\mathcal{J}}$ is computed as (using one-indexing):

$$\frac{1}{2}\max|\mathbf{w}| = 3$$
$$\mathcal{J}_{large} = \{j \in \mathcal{J} \mid |w_j| \leq \frac{1}{2}\max|\mathbf{w}|\} = \{1, 2\}$$

$\mathcal{J} \setminus \bar{\mathcal{J}} = \{3\}$ is the set of indices of where each index in the set can form a singleton inclusion set. So there is one inclusion set from $\mathcal{J} \setminus \bar{\mathcal{J}}$, $\mathcal{O}_1^* = \{3\}$. And from $\bar{\mathcal{J}}$, there is only one optimal inclusion set $\mathcal{O}_2^* = \{1, 2\}$. So the optimal inclusion sets are:

$$\mathcal{O}_1^* = \{3\} \qquad \sum_{j \in \mathcal{O}_1^*} |w_j| = 6 > 3$$
$$\mathcal{O}_2^* = \{1, 2\} \qquad \sum_{j \in \mathcal{O}_2^*} |w_j| = 3.5 > 3$$

The corresponding split weight vectors are computed as:

$$\mathcal{O}_1^* = \{3\} \qquad \bar{\mathbf{w}}_1 = [0, 0, -6]$$
$$\mathcal{O}_2^* = \{1, 2\} \qquad \bar{\mathbf{w}}_2 = [6, 6, 0]$$

And the translated rules are:

$$\mathfrak{L}_{\mathcal{W}^\star} = \left\{ \begin{array}{l} \mathrm{c} \leftarrow \mathtt{not}\ \mathrm{a}_3. \\ \mathrm{c} \leftarrow \mathrm{a}_1, \mathrm{a}_2. \end{array} \right\}.$$

This rule set matches the minimal rule set $\mathfrak{L}$ in Equation (7.20) above.

Algorithm 6 The full algorithm on splitting a negatively used conjunctive node with optimisation

Input: Weight $\mathbf{w}$ of a conjunctive node negatively connected to a disjunctive node
Output: List of valid split weights

Compute $\mathcal{J} \leftarrow \{j \in \{1..s\} | w_j \neq 0\}$
Compute $\mathcal{J}_{large} \leftarrow \{j \in \mathcal{J} \mid |w_j| > \max|\mathbf{w}|/2\}$
Compute $\mathcal{J}_{small} \leftarrow \{j \in \mathcal{J} \mid |w_j| \leq \max|\mathbf{w}|/2\}$
if $|\mathcal{J}_{large}| = 0$ **then**
    /* Every element in the original node's weight vector is needed. */
    return $[\mathrm{sign}(\mathbf{w}) \cdot 6]$
overall_valid_splits $\leftarrow \{\}$
/* Add all large weights as singleton sign-mismatch sets */
for $j \in \mathcal{J}_{large}$ **do**
    overall_valid_splits.add($\{j\}$)
/* Explore combinations of small weights that exceed threshold */
accumulators $\leftarrow [\{j\} \mid j \in \mathcal{J}_{small}]$
while *accumulators* is not empty **do**
    Pop the head item *candidate_set* from *accumulators*
    sum_weights $\leftarrow \sum_{j \in candidate_set} |w_j|$
    if *sum_weights* $> \max|\mathbf{w}|/2$ **then**
        /* This is now a valid inclusion set. */
        /* Check if it is subsumed by any other valid inclusion set. */
        /* If not, then add it to *overall_valid_splits*. */
        if *candidate_set* is not subsumed **then**
            overall_valid_splits.add(*candidate_set*)
    else
        /* Continue exploring larger combinations */
        for $j \in \mathcal{J}_{small}$ and $j \notin$ *candidate_set* **do**
            new_candidate $\leftarrow$ sort(*candidate_set* $\cup \{j\}$)
            accumulators.append(*new_candidate*)
split_tensors $\leftarrow []$
for each valid sign-mismatch set $\mathcal{S} \in$ *overall_valid_splits* **do**
    Create new tensor $\tilde{\mathbf{w}}$ where:
      - $j \in \mathcal{S}$: $\tilde{w}_j = -6 \cdot \mathrm{sign}(w_j)$
      - $j \in \mathcal{J} \setminus \mathcal{S}$: $\tilde{w}_j = 0$
    Add $\tilde{\mathbf{w}}$ to *split_tensors*
return *split_tensors*

7.3.4 Model Sparsification

Even with the optimisation process, the computation of candidate indices can still become an expensive combinatorial problem with the worst-case complexity $O(2^s)$ when the number of inputs s to a conjunctive node is large. Note that this s might be different to the original number of inputs s' from the task since this disentanglement process operates on a pruned neural DNF-based model. Since the model may not be sparse enough after pruning, further sparsification is required at the training stage. Top-K is an operation that can be used either as an activation function in sparse autoencoders [100, 58], where only the k largest latents are kept and the rest are set to 0, or used as a randomly-initialised weight mask [76] to explore a subset of parameters with large magnitude. This work takes inspiration from Top-K to further sparsify the model at the model design stage, by only connecting a conjunctive node to a randomly-selected subset of k inputs, and setting the connection to the rest of the inputs to 0 throughout training. Different from Top-KAST [76], this approach does not have an exploration loss to change the weight mask, which could be a future improvement. Furthermore, L_0 approximation-based approaches [98, 135] may also be suitable for this problem.

7.4 Additional Learning Techniques

This section introduces two additional learning techniques that are not directly related to the disentanglement method but are used in the experiments of this chapter. Specifically, Section 7.4.1 introduces the predicate invention method, and Section 7.4.2 introduces the improved delta scheduling method.

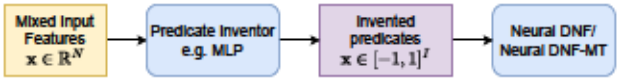
7.4.1 Predicate Invention

Since the inputs of the semi-symbolic nodes are restricted to the range $[-1, 1]$ to support neuro-symbolic interpretation, neural DNF-based models require **predicate invention** to handle non-bivalent inputs. Both pix2rule [28] and the work presented in Chapter 6 use CNNs to extract higher-level features as predicates from low-level unstructured inputs while maintaining the ability to train the whole architecture end-to-end. The architecture design of the predicate inventor is not limited to CNNs but can include any neural models such as MLPs or even transformers. However, these neural predicates are not inherently interpretable, and the attempt in Chapter 6 to improve the interpretability through an ASP optimisation process does not guarantee a solution. In principle, the neural-network-based predicate inventor can be distilled into a neural DNF-based model that can be translated into some form of logical representation. The distillation in the work presented in Section 6.3.3 is to train a neural DNF-based model that outputs the same as the MLP under the same input. This chapter does not use the distillation method as it focuses on an inherently interpretable threshold-learning predicate invention that will be discussed now.

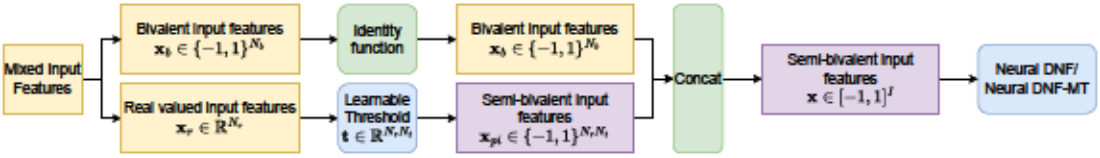
This section introduces an inherently interpretable approach called **threshold-learning predicate invention** (Figure 7.5c). For each real-valued input feature x_i, the method learns m threshold values $\{t_{i,1}, \dots t_{i,m}\}$, which are each used to create a predicate $p_{i,j} = \tanh((x_i - t_{i,j})/T)$ where $T \in [0.1, 1]$ is a temperature parameter. The invented predicate $p_{i,j}$ can be interpreted as a bivalent predicate representing the inequality $x_i > t_{i,j}$ if $p_{i,j} > 0$. The method schedules the temperature



(a) No predicate invention.



(b) Neural-network-based predicate invention. The architecture is not restricted to MLPs only but can be any neural models such as CNNs or even transformers.



(c) The novel threshold-learning predicate invention, which can be translated to inequalities and thus provide interpretability.

Figure 7.5: Architectures for various settings with different types of inputs, where blue rounded boxes have learnable parameters. 7.5a is the basic setup for bivalent inputs. 7.5b and 7.5c show how predicate invention is performed with MLPs and threshold learning.

T to decrease throughout training in the same way as the exponential delayed decay of δ, in order to push $p_{t,j}$ away from 0, thus amplifying the degree of belief in the invented predicate. This threshold-learning predicate invention does not need post-training methods to be interpretable and guarantees a solution.

7.4.2 Improved Delta Scheduling

As mentioned in Section 5.2.2, and possibly Section 7.2.2, the change in the magnitude of δ throughout training can heavily affect the learning result of the neural DNF-based model, since the stricter the logical bias becomes, the more difficult it is for the model to learn. This section introduces an improved delta scheduling method that can mitigate the issue.

Delayed Decay Delta Scheduling with Different Update Functions

Recall the exponential decay scheduler defined in Equation 2.7 in Section 2.3.2:

$$|\delta_{t+1}| = \begin{cases} \delta_0 & \text{if } t < t_{\text{delay}} \\ \min\left(\delta_0 \cdot r^{\left\lfloor \frac{t - t_{\text{delay}}}{t_{\text{steps}}} \right\rfloor + 1}, 1\right) & \text{if } t \geq t_{\text{delay}} \end{cases}$$

where $\delta_0 \in (0,1)$ is the initial magnitude of δ, $r > 1$ is the decay rate, $t_{\text{delay}} \in \mathbb{N}$ is the delay epoch before the magnitude of δ starts to increase, and $t_{\text{steps}} \in \mathbb{N}_{>0}$ is the number of epochs between updates. Note that the absolute value is used as the sign of δ is positive for conjunctive nodes and negative for disjunctive nodes.

The delayed decay scheduler can be generalised to the following:

$$|\delta_{t+1}| = f_{\delta,\text{delayed}}(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r, g) = \begin{cases} \delta_0 & \text{if } t < t_{\text{delay}} \\ \min\left(g(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r), 1\right) & \text{if } t \geq t_{\text{delay}} \end{cases} \tag{7.21}$$

where $g(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r)$ is an update function that computes the new magnitude of δ at time t. The exponential decay scheduler above uses the following update function:

$$g_{\text{exponential}}(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r) = \delta_0 \cdot r^{\left\lfloor \frac{t - t_{\text{delay}}}{t_{\text{steps}}} \right\rfloor + 1} \tag{7.22}$$

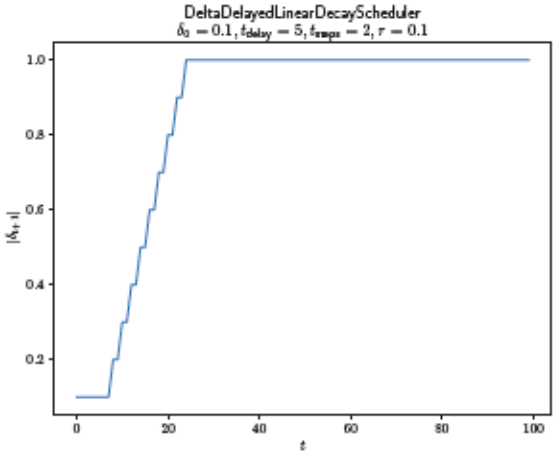
Apart from the exponential update function, this section introduces the following update functions:

$$g_{\text{linear}}(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r) = \delta_0 + r \cdot \left\lfloor \frac{t - t_{\text{delay}}}{t_{\text{steps}}} \right\rfloor, r \in (0, 1) \tag{7.23}$$

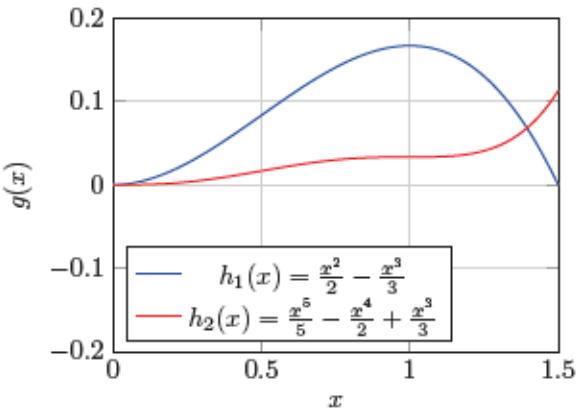
$$\begin{gathered} g_{\text{monotonic},h_i}(t, t_{\text{delay}}, t_{\text{steps}}, r) = \frac{h_i(\xi)}{h_i(1)} \\ \text{where } r > 1,\ i \in \{1, 2\},\ \xi = \min\left(\frac{\left\lfloor \frac{t - t_{\text{delay}}}{t_{\text{steps}}} \right\rfloor}{r}, 1 \right), \text{ and} \\ h_1(x) = \frac{x^2}{2} - \frac{x^3}{3} \\ h_2(x) = \frac{x^5}{5} - \frac{x^4}{2} + \frac{x^3}{3} \end{gathered} \tag{7.24}$$

Note that δ_0 is not used for the monotonic update functions and thus not included in the theoretical definition. However, it is kept in the code for implementation consistency and is ignored in the calculations.

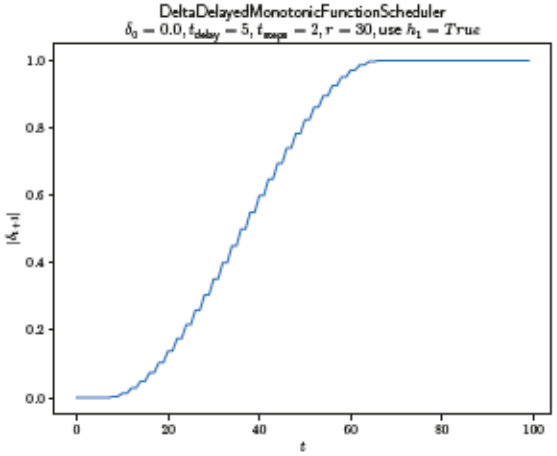
Figure 7.6 shows examples of the new delayed delta scheduling methods with different update functions. The new update functions show slower growth when $|\delta_{t+1}|$ is close to 1 compared to the exponential update function, which is the biggest obstacle for the model to learn as the logical bias approaches full strength.



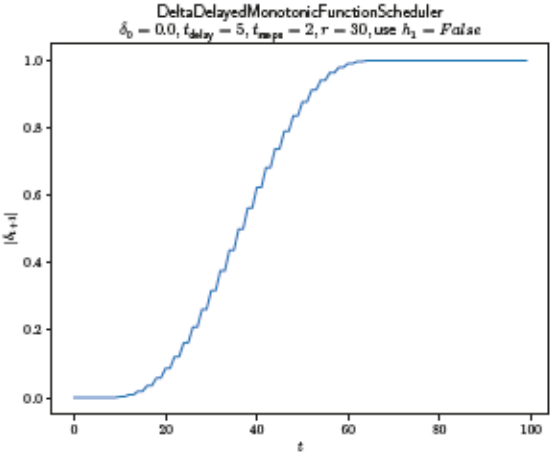
(a) An example of the delayed linear delta scheduling, using $g_{\text{linear}}(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r)$ in Equation (7.23) with parameters $\delta_0 = 0.1$, $r = 0.1$, $t_{\text{delay}} = 5$ and $t_{\text{steps}} = 2$.



(b) Plot of $h_1(x)$ and $h_2(x)$ used in the monotonic update function $g_{\text{monotonic},h_i}(t, t_{\text{delay}}, t_{\text{steps}}, \delta_0, r)$. These two functions are chosen for their slow monotonic increase from $x \in [0.5, 1]$ range.



(c) An example of delayed monotonic scheduling using $g_{\text{monotonic},h_1}(t, t_{\text{delay}}, t_{\text{steps}}, r)$ in Equation (7.24) with parameters $r = 0.1$, $t_{\text{delay}} = 5$ and $t_{\text{steps}} = 2$. The initial value of δ is always 0 and hyperparameter δ_0 is not used for this function, but kept for implementation consistency.



(d) An example of delayed monotonic scheduling using $g_{\text{monotonic},h_2}(t, t_{\text{delay}}, t_{\text{steps}}, r)$ in Equation (7.24) with parameters $r = 0.1$, $t_{\text{delay}} = 5$ and $t_{\text{steps}} = 2$. The initial value of δ is always 0 and hyperparameter δ_0 is not used for this function, but kept for implementation consistency.

Figure 7.6: Examples of new delayed delta scheduling methods with different update functions and different settings.

Delayed Performance-monitoring Delta Scheduling

Apart from using t_{steps} to control the update frequency, the scheduler can replace the waiting steps with a check on the model's performance every epoch against the train set, and only update the magnitude of δ if the performance reaches a desired target or is close to it within a tolerance. Note that the magnitude of δ is not guaranteed to reach 1 under this setting, since if the performance does not reach the desired target, the magnitude of δ will not be updated. However, if the update does occur, the learning is more stable. Using i_{t+1} as an internal counter initialised to 0, the delayed monitoring exponential and linear schedulers are defined as follows:

$$f_{\delta,\text{delayed monitoring exponential}}(t, t_{\text{delay}}, \delta_0, r, \epsilon_{\text{perf}}, i_{t+1}) = \min\left(\delta_0 \cdot r^{i_{t+1}}, 1\right) \tag{7.25}$$

$$f_{\delta,\text{delayed monitoring linear}}(t, t_{\text{delay}}, \delta_0, r, \epsilon_{\text{perf}}, i_{t+1}) = \min\left(\delta_0 + r \cdot i_{t+1}, 1\right) \tag{7.26}$$

$$i_{t+1} = \texttt{computeInternalCounter}(t, t_{\text{delay}}, i_t, \text{current_perf}, \text{order}, \epsilon_{\text{perf}}) \tag{7.27}$$

`computeInternalCounter` is defined as follows:

Algorithm 7 Internal step counter for delayed monitoring delta scheduling

function COMPUTEINTERNALCOUNTER($t, t_{\text{delay}}, i_t, \text{current_perf}, \text{order}, \epsilon_{\text{perf}}$)
▷ Interal target performance tracker *internal_target_perf* needs to be stored in persistent memory that can be accessed across each function call
  if $t < t_{\text{delay}}$ **then**
    return 0
  if $t = t_{\text{delay}}$ **then**
    internal_target_perf ← current_perf ▷ Set target perf. for the first time
  if order = "ascending" **and** current_perf − internal_target_perf $< -\epsilon_{\text{perf}}$ **then**
    return i_t ▷ Perf. below target, return current counter
  if order = "descending" **and** current_perf − internal_target_perf $> -\epsilon_{\text{perf}}$ **then**
    return i_t ▷ Perf. above target, return current counter
  internal_target_perf ← current_perf
  return $i_t + 1$

Figure 7.7 shows an example of the delayed monitoring linear delta scheduling used in training.

7.5 Experiments

7.5.1 Setup

This section trains and evaluates the neural DNF-based models across *binary*, *multi-class* and *multi-label* classification tasks, including tasks with real-valued features that require *predicate inventions*:

- Binary: BCC† [122], Monk [162], Mushroom [67] and CDC† [152].
- Multi-class: Car [22] and Covertype‡ [21].
- Multi-label: ARA [25], Budding [94], Fission [35], and MAM [52].

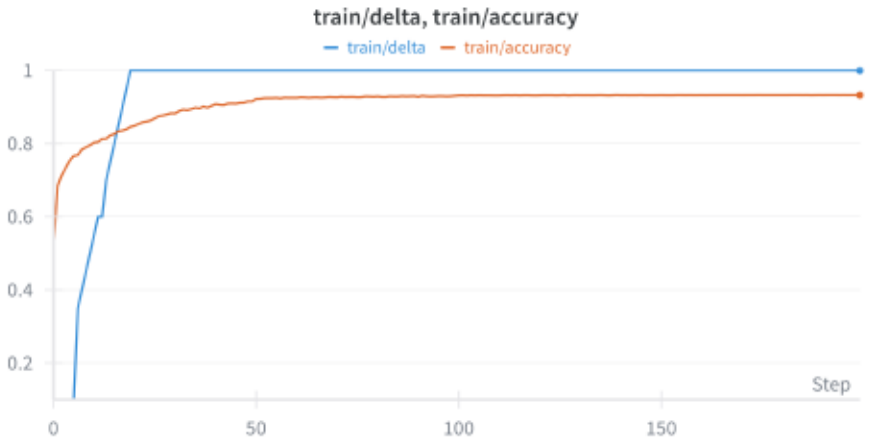
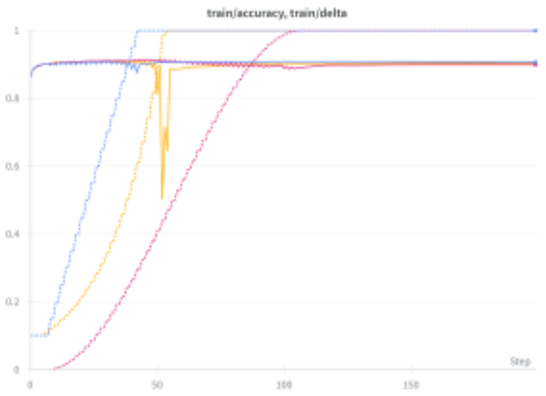


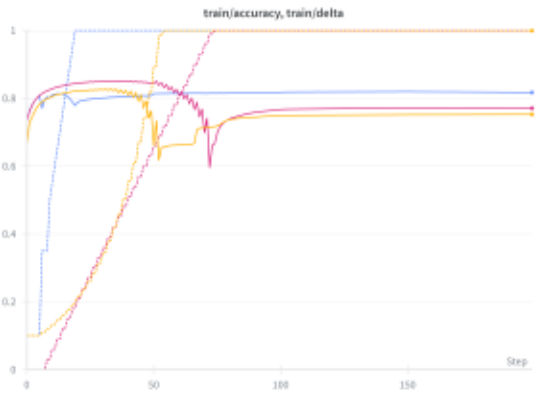
Figure 7.7: An example of the delayed monitoring linear delta scheduling using $f_{\delta,\text{delayed monitoring linear}}$ from Equation 7.26. The schedule halts at $|\delta| = 0.6$ before continuing to increase.

The multi-label datasets are derived from Boolean Networks [82], which model the state transitions of genes from time t to $t+1$. Learning such dynamic systems is formulated as a multi-label classification problem, where the target labels correspond to the gene states at time $t+1$. These datasets are accompanied by ground-truth logic programs, enabling direct evaluation of the extracted rules. Datasets marked with † and ‡ contain real-valued features that necessitate predicate invention for the neural DNF-based models: threshold-learning predicate invention is employed for the former, and MLP predicate invention for the latter. Regarding model selection, neural DNF is applied to binary and multi-label tasks, whereas neural DNF-MT and neural DNF-EO are employed for multi-class classification. Exponential updates are adopted as the default choice for most experiments, as they provide a simple yet effective approach with minimal performance variation across alternative update functions. Two exceptions are the CDC and Covertype datasets, which employ monotonic and monitoring linear updates, respectively, to better suit their specific characteristics. Figure 7.8 compares the performance of different delayed delta scheduling methods during training on the CDC and Covertype datasets.

The experiments consider machine learning models that do not rely on task-specific background knowledge as baselines, including: Logistic Regression, Random Forest, Decision Tree, Support Vector Machines (SVMs) and MLPs. While these models offer varying degrees of interpretability, decision trees and neural DNF-based models stand out for their high interpretability by providing conditional prediction in a rule-based decision-making process: decision trees provide decision paths that can be followed from inputs to decisions; and the neural DNF-based models are designed such that they provide rule-based interpretations. Logistic regression and random forests are two less interpretable models with a compositional nature: logistic regression shows the additive contribution of inputs to the log-odds of the final decision; each tree in a random forest is rule-based but the overall decision is derived from a weighted voting system of the trees' decisions. SVMs and MLPs are typically not considered interpretable due to their complex decision boundaries and lack of transparency in decision-making processes. Because of the different characteristics, this study focuses on decision trees as direct competitors to neural DNF-based models for the interpretability study,



(a) Training of neural DNF model on the CDC dataset using different delayed monitoring delta scheduling methods, with exponential update function, linear update function and monotonic update function h_1. The exponential update shows more turbulent changes in the accuracy curve compared to the linear and monotonic updates.



(b) Training of neural DNF-MT model on the Covertype dataset using different delayed monitoring delta scheduling methods, with exponential update function, monitoring linear update function and linear update function. The monitoring linear update shows more a stable training process with little accuracy degradation compared to the exponential and linear updates.

Figure 7.8: Comparison of different delayed delta scheduling methods during training on the CDC and Covertype datasets. The accuracy curves are shown as solid lines and the delta magnitude curves are shown as dashed lines. The figures show the stability benefits of different update functions over exponential updates.

and ignores the other models with non-rule-based interpretations. Optimal Sparse Decision Tree (OSDT) [71] that addresses tree complexity is also included as a better competitor, but OSDT is limited to binary features and lacks support for multi-class classification.

7.5.2 Results

The first rows in Table 7.1, 7.2 and 7.4 show that neural DNF-based models achieve competitive F1 scores compared to standard ML approaches that also do not rely on task-specific background knowledge, with neural DNF-based models performing the best in 7 of the 10 datasets. The neural DNF-based models match the level of expressiveness while enabling end-to-end training with predicate invention for real-valued inputs.

The second rows of Table 7.1, 7.2 and 7.4 present the performance of the novel optimised logical interpretation of the models as logic programs (NDNF: ASP and NDNF-MT: LI resp.), which in most cases lead to a minor performance drop compared to the neural DNF-based models but at the benefit of interpretability. Table 7.5 shows that the new interpretation method has a much smaller reduction in performance (right column) compared with the pre-existing threshold method (left column) in all but one dataset (Covertype). This exception may relate to the lack of faithfulness guarantee for conjunctive node disentanglement when using predicate invention, along with a higher task complexity.

The results also show that the neural DNF-EO models perform worse than the neural DNF-MT model on the same tasks. Table 7.3 provides the additional Jaccard index scores for the neural DNF-EO models in multi-class tasks. The use of Jaccard index scores is to measure both the

correctness and the mutual exclusivity of the model's output. The closer to 1, the better the model's performance. Although the neural DNF-EO model can perform well before any post-training process, displaying high F1 and Jaccard index scores, the ASP interpretation of the model's output drops significantly, especially for the Covertype dataset. The neural DNF-EO model's constraint layer (the last layer, see Figure 5.1) enforces the logical mutual exclusivity constraint. The performance drop after post-training logical interpretation shows that the learned neural representation sets at an unstable equilibrium that cannot be maintained after modifications on the model during the post-training process, reaching a similar conclusion to the brief discussion in Section 5.3.2. In comparison, the neural DNF-MT model enforces the probabilistic mutual exclusivity constraint via its mutex-tanh activation in the disjunctive layer and the disjunctive layer is thus translated into probabilistic logic instead of bivalent logic. The mutex-tanh activation allows more flexibility in learning via the softmax function and the probabilistic interpretation does not modify the disjunctive layer, better preserving the model's internal representation.

To assess the compactness of the logical interpretations induced by the disentangled neural DNF-based models, this section compares the length of the rules of its ASP program (full ASP programs for binary/multi-label tasks, ASP programs for conjunctive layers in the neural DNF-MT models in multi-class tasks) with the tree sizes of decision trees and OSDTs trained on the same datasets. For a decision tree's decision path, each node in the path can be seen as a condition clause of an if statement, thus making the decision tree's depth an equivalent metric to ASP rule length. The number of branches indicates the number of different decision paths, making it equivalent to the number of rules in an ASP program. Table 7.6 shows the models' interpretations are consistently more compact than those of decision trees, as shown by them having fewer rules and both the average and maximum rule length being lower. Compared to OSDT, the models have comparable compactness (they are more compact for the Monk dataset while OSDT is more compact for the Mushroom dataset) whilst achieving superior performance. Listing 7.1 shows the ASP interpretation of a neural DNF model in the MONK dataset.

To summarise, the neural DNF-based models demonstrate comparable performance to other methods that also do not rely on background knowledge and train well together with end-to-end predicate invention, while also providing compact logical representations at a minimal performance cost. In particular, the new disentanglement approach reduces such performance loss when discretising the neural DNF-based models compared to the previous thresholding method.

```
% t is the target class predicate.
% a_i is the input feature.
t :- a_1, a_4.    t :- a_2, a_5.    t :- a_0, a_3.
t :- not a_11, not a_12, not a_13.
```

Listing 7.1: ASP interpretation of a neural DNF model in the MONK dataset. More examples can be found in Appendix B.3.

7.6 Discussions

The experimental results demonstrate that the proposed disentanglement method substantially reduces performance degradation when translating neural DNF-based models into bivalent logic rep-

Model	Monk	BCC†	Mushroom	CDC†
NDNF	**1.000 ± 0.000**	0.718 ± 0.072	**1.000 ± 0.000**	0.882 ± 0.001
NDNF: ASP	0.992 ± 0.008	0.765 ± 0.033	0.971 ± 0.016	0.861 ± 0.003
MLP	**1.000 ± 0.000**	0.725 ± 0.025	0.999 ± 0.000	**0.896 ± 0.000**
Logistic Regression	0.707 ± 0.006	**0.796 ± 0.020**	**1.000 ± 0.000**	0.873 ± 0.000
SVM	**1.000 ± 0.000**	0.762 ± 0.029	**1.000 ± 0.000**	0.894 ± 0.000
Random Forest	0.984 ± 0.005	0.753 ± 0.012	**1.000 ± 0.000**	0.889 ± 0.000
Decision Tree	0.877 ± 0.027	0.766 ± 0.030	**1.000 ± 0.000**	0.871 ± 0.002
OSDT	**1.000 ± 0.000**	NA	0.893 ± 0.000	NA

Table 7.1: F1 scores (mean ± ste) for binary classification tasks. 'NDNF: ASP' is the logical interpretation of neural DNF-based models. F1 score is not measured for neural DNF-EO's ASP interpretation as there might be multiple classes included in the answer set and thus violating the exactly-one constraint of multi-class classification.

Model	Car	Covertype‡
NDNF-MT	0.963 ± 0.001	0.859 ± 0.003
NDNF-MT: LI	0.935 ± 0.005	0.823 ± 0.003
NDNF-EO	**0.973 ± 0.001**	0.813 ± 0.005
MLP	0.869 ± 0.005	0.828 ± 0.001
Logistic Regression	0.891 ± 0.003	0.758 ± 0.000
SVM	0.937 ± 0.004	0.716 ± 0.001
Random Forest	0.954 ± 0.002	**0.942 ± 0.000**
Decision Tree	0.959 ± 0.004	0.914 ± 0.002

Table 7.2: F1 scores (mean ± ste) for multi-class tasks. 'NDNF-MT: LI' is the logical interpretation of neural DNF-MT models.

Model	Car	Covertype‡
NDNF-EO	0.885 ± 0.003	0.630 ± 0.005
NDNF-EO: ASP	0.842 ± 0.012	0.362 ± 0.043

Table 7.3: Jaccard index scores (mean ± ste) for Neural DNF-EO models in multi-class tasks. 'NDNF-EO: ASP' is the logical interpretation of neural DNF-EO models.

Model	ARA	Budding	Fission	MAM
NDNF	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
NDNF: ASP	0.993 ± 0.002	0.993 ± 0.003	0.987 ± 0.003	0.994 ± 0.003
MLP	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**	**1.000 ± 0.000**
Random Forest	**1.000 ± 0.000**	0.996 ± 0.000	0.992 ± 0.001	0.998 ± 0.001
Decision Tree	**1.000 ± 0.000**	0.972 ± 0.001	0.954 ± 0.002	0.950 ± 0.002

Table 7.4: F1 scores (mean ± ste) for multi-label tasks. 'NDNF: ASP' is the logical interpretation of neural DNF-based models.

	Dataset	$f1_{train} - f1_{thresh}$	$f1_{train} - f1_{disent}$
Bin.	Monk	0.076 ± 0.027	$\mathbf{0.008 \pm 0.008}$
	BCC†	0.050 ± 0.038	$\mathbf{0.003 \pm 0.022}$
	Mushroom	0.316 ± 0.099	$\mathbf{0.029 \pm 0.016}$
	CDC†	0.031 ± 0.007	$\mathbf{0.021 \pm 0.003}$
MC.	Car	0.122 ± 0.012	$\mathbf{0.027 \pm 0.005}$
	Covertype‡	$\mathbf{0.036 \pm 0.003}$	0.043 ± 0.004
ML.	ARA	0.085 ± 0.009	$\mathbf{0.007 \pm 0.002}$
	Budd	0.159 ± 0.004	$\mathbf{0.007 \pm 0.003}$
	Fission	0.141 ± 0.007	$\mathbf{0.013 \pm 0.003}$
	MAM	0.045 ± 0.004	$\mathbf{0.005 \pm 0.003}$

Table 7.5: F1 score drops after thresholding ($f1_t - f1_{thresh}$) and disentanglement ($f1_t - f1_{disent}$) discretisation.

	Dataset	Max. Rule Length/Max Depth NDNF/ NDNF-MT: ASP	Max. Rule Length/Max Depth DT/ *OSDT	Avg. Rule Length/Avg Depth NDNF/ NDNF-MT: ASP	Avg. Rule Length/Avg Depth DT/ *OSDT	No. Rules/No. Branches NDNF/ NDNF-MT: ASP	No. Rules/No. Branches DT/ *OSDT
Bin.	Monk	$\mathbf{2.778 \pm 0.148}$	10.000 ± 0.283 $*4.000 \pm 0.000$	$\mathbf{2.042 \pm 0.053}$	7.842 ± 0.098 $*3.286 \pm 0.000$	$\mathbf{4.000 \pm 0.000}$	49.000 ± 7.071 $*7.000 \pm 0.000$
	BCC†	$\mathbf{3.200 \pm 0.179}$	6.600 ± 0.219	$\mathbf{2.267 \pm 0.121}$	4.833 ± 0.062	$\mathbf{5.000 \pm 0.566}$	18.200 ± 0.657
	Mush	4.250 ± 0.225	7.400 ± 0.358 $*\mathbf{2.400 \pm 0.219}$	2.982 ± 0.122	4.907 ± 0.132 $*\mathbf{1.900 \pm 0.128}$	3.750 ± 0.242	18.000 ± 1.095 $*\mathbf{3.400 \pm 0.219}$
	CDC†	$\mathbf{4.800 \pm 0.276}$	40.600 ± 0.607	$\mathbf{2.021 \pm 0.076}$	21.804 ± 0.021	$\mathbf{27.600 \pm 0.851}$	31394 ± 257.045
MC.	Car	$\mathbf{8.500 \pm 0.153}$	13.000 ± 0.283	$\mathbf{4.432 \pm 0.075}$	9.462 ± 0.060	$\mathbf{25.688 \pm 0.943}$	109.200 ± 1.927
	Cov‡	$\mathbf{4.300 \pm 0.202}$	25.200 ± 0.657	$\mathbf{1.778 \pm 0.041}$	14.541 ± 0.047	$\mathbf{38.900 \pm 1.090}$	868.400 ± 8.083
ML.	ARA	$\mathbf{3.900 \pm 0.095}$	13.000 ± 0.000	$\mathbf{2.050 \pm 0.024}$	9.724 ± 0.006	$\mathbf{26.700 \pm 0.318}$	581.800 ± 2.125
	Bud	$\mathbf{4.000 \pm 0.000}$	12.000 ± 0.000	$\mathbf{3.126 \pm 0.036}$	10.937 ± 0.007	$\mathbf{48.400 \pm 1.070}$	1521.300 ± 6.406
	Fis	$\mathbf{6.000 \pm 0.000}$	10.000 ± 0.000	$\mathbf{2.852 \pm 0.020}$	9.308 ± 0.018	$\mathbf{21.800 \pm 0.276}$	392.500 ± 4.401
	MAM	$\mathbf{4.000 \pm 0.000}$	10.000 ± 0.000	$\mathbf{2.645 \pm 0.014}$	0.897 ± 0.018	$\mathbf{21.500 \pm 0.212}$	424.200 ± 5.261

Table 7.6: Comparing compactness of neural DNF-based models' logic interpretations vs. decision trees', i.e. max. rule length/tree depth, avg. rule length/tree depth, and no. rules/branches. The lower the number the better the compactness. The cell marked with * refers to OSDT's performance. OSDT is only applicable when the input features are boolean and the target is a single class.

resentations, outperforming the conventional thresholding approach across binary, multi-class, and multi-label classification tasks. The method successfully addresses the entanglement problem at the conjunctive layer by splitting entangled nodes into multiple smaller discretised nodes that collectively preserve the original behaviour. However, there remain certain scenarios where the disentanglement method faces challenges. This section examines two key considerations: the limitations that arise when entanglements involve disjunctive layers, and the computational scalability of the disentanglement process.

7.6.1 Limitations when Disentangling Disjunctive Layers

As seen in the experiments, the disentanglement method works well in most cases. However, it cannot handle certain cases where the entanglements involving both conjunctive and disjunctive nodes lead to imperfect disentanglement. Example 7.6.1 shows a case where disjunctive nodes assign weights to utilise small-magnitude activations of conjunctive nodes to match the target. As discussed in Section 7.3, the disentanglement requires node inputs to be exactly $\{-1, 1\}^N$. Despite using an auxiliary loss to encourage ± 1 conjunctive activations, this constraint is not guaranteed for the disjunctive nodes. The activations of conjunctive nodes are not close to ± 1 and disjunctive nodes assign weights accordingly to utilise conjunctive nodes' activations with small magnitude to match the target training data. Such non-extreme activations in conjunctive nodes prevent applying the same disentanglement method to the disjunctive layer, and experiments show that splitting disjunctive nodes performs worse than simply thresholding. Future work can investigate how to avoid these intermediate-valued cases.

Example 7.6.1: Nested Entanglement

Consider two conjunctive nodes with weights $\mathbf{w}_{c1} = [-1.33, 0, 0, 1.01, -1.44]$ and $\mathbf{w}_{c2} = [-2.02, -0.79, -0.79, 0.71, -1.52]$ respectively; and two disjunctive nodes connected to those conjunctive nodes, with weights $\mathbf{w}_{d1} = [3.43, 1.28]$ and $\mathbf{w}_{d2} = [0, 3.56]$ respectively. There is a nested entanglement among all the nodes here: $c1$ subsumes $c2$, and $d1$ uses a combination of $c1$ and $c2$ to match the target label. The disjunctive nodes' bivalent interpretations change depending on whether the input activations from the conjunctive nodes are soft-valued or strictly ± 1, even when the interpretations of the conjunctive nodes' activations are the same (differences marked in red in the table below).

a1	a2	a3	a4	a5	c1	c2	d1	d2
-1	-1	-1	-1	-1	-0.52/⊥	0.54/⊤	0.17/⊤	1.00/⊤
		—			-1/⊥	1/⊤	-0.70/⊥	1.00/⊤
-1	-1	-1	1	1	-0.89/⊥	-0.77/⊥	-0.99/⊥	0.67/⊤
		—			-1/⊥	-1/⊥	-1.00/⊥	0.00/⊥

7.6.2 Scalability of Disentanglement

The disentanglement method's runtime grows exponentially with the number of inputs used in a node, as shown in Figure 7.9, since it may check all possible weight combinations that trigger the

node to fire for the worst-case scenario. This makes it less scalable than the thresholding discretisation method, which scales linearly with weight magnitudes. However, this limitation only affects model interpretation, not learning capability, and can be mitigated through model sparsification (Section 7.3.4).

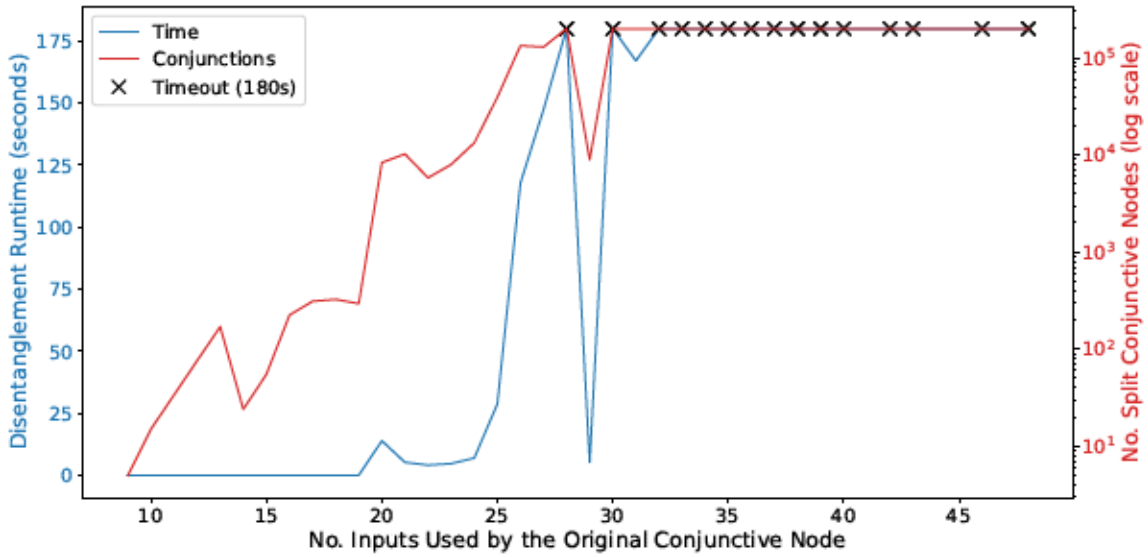


Figure 7.9: Time taken (blue) to disentangle the conjunctive nodes with different numbers of inputs used, and the number of split conjunctions (red) after disentanglement, in a trained neural DNF model. A max runtime of 180s is set for disentanglement.

7.7 Conclusion

This chapter proposes a disentanglement discretisation method that enables more faithful translation of neural DNF-based models into bivalent logic representations. Additionally, it introduces an interpretable-by-design predicate invention method that learns threshold values end-to-end, effectively converting real-valued features into inequalities as boolean predicates. The delta scheduling method is also improved to reduce the destructive effect of the forced logical bias on the learning of the neural DNF-based models that leads to poor performance. The experiments across various real-world classification tasks demonstrate that the disentanglement method provides compact and interpretable logical representations for the neural DNF-based models without requiring task-specific background knowledge, while improving performance over the existing thresholding method.

Chapter 8

Conclusion and Future Work

This thesis has presented a family of neural DNF-based models that can search in large continuous hypothesis spaces without prior knowledge and provide compact and interpretable decision rules for human readability and curation. Having examined the theoretical foundations, proposed novel architectures for classification and reinforcement learning, and addressed the challenge of discretisation, this chapter reflects on the contributions made and outlines promising directions for future research.

8.1 Summary of Contributions

This thesis provides a number of contributions.

The first fundamental work is the formalisation and proof of key properties of neural DNF-based models, which support the logical interpretation and intervention on the models. Chapter 4 formalises the semi-symbolic node's behaviour and interpretation, and proves the crucial property that the bivalent logic-neural translation is bidirectional and maintains the truth value equivalence, given appropriate constraints. The bivalent-logic-to-neural translation is fundamental for the interpretation of the trained neural DNF-based models, while the neural-to-bivalent-logic translation provides a way to intervene on the model.

The *main objective* of this thesis is to propose a family of neural DNF-based models that perform differentiable rule learning for classification and reinforcement learning tasks. Inspired by the semi-symbolic node formulation in pix2rule [28], this thesis develops novel architectures tailored to the distinct requirements of each domain. Chapter 5 proposes the neural DNF-EO model for deterministic multi-class classification tasks and investigates how to adopt the logical mutual-exclusive constraint on the outputs from a neural DNF-based architecture. Chapter 6 proposes the neural DNF-MT model for probabilistic policy learning in reinforcement learning settings. Different from the logical mutual-exclusive constraint in the neural DNF-EO model, the neural DNF-MT model focuses on the probabilistic mutual-exclusive constraint and provides a probabilistic interpretation of the model's behaviour. Chapter 6 also demonstrates the intervention capability of the model by policy modification.

The *main challenge* throughout the thesis is to *discretise* the weights of a neural DNF-based model while preserving the model's behaviour. Chapter 7 investigates the performance degradation

of the neural DNF-based models caused by the conversion of the model's learned hypothesis from a continuous space to a discrete space for logical interpretation. It proposes a disentanglement method that splits the semi-symbolic nodes to multiple nodes in order to preserve the same logical interpretation, thus preserving the model's behaviour.

8.2 Future Work

The two main directions of future work are (i) to further explore the neural DNF-based model in its theoretical properties, architecture and training regime, and (ii) to apply the neural DNF-based models to other real-life domains.

8.2.1 Model Extensions

The ability to translate between bivalent logic and neural representations opens further opportunities. In standard practice, a neural network is initialised with random weights before training. However, a neural DNF-MT actor can instead be warm-started: an existing policy, either already in neural form or translated from ASP form, can be used as initial network weights. This approach is particularly useful when deploying the actor in a similar but previously unseen environment. Suppose a neural DNF-MT actor has been trained in environment A; the actor can be retrained in environment B with some similarities to environment A, thus mimicking the behaviour of transferring background knowledge from environment A to B. The bivalent-logic-to-neural translation can also be used as a form of background knowledge injection. A neural DNF-based model may encode expected partial behaviours of the task, if available before training, to further speed up the training process.

A second direction is to examine whether the interpretable structure of a neural DNF-based model propagates to its neural predicate inventor during end-to-end training. Figure 8.2 shows the CAMs of the CNN encoder connected to an MLP and a neural DNF-MT actor in the Door Corridor environment. The CAMs are generated using Grad-CAM [139] and EigenCAM [111], and show the regions of the input that the actor has focused on to make its decision. The CAMs of the CNN connected to the neural DNF-MT actor show different regions of focus compared to the MLP actor. For some steps, the CNN's region of focus is aligned to the interpreted logical policy. This preliminary observation raises the hypothesis that the neural DNF-MT actor may improve the interpretability of the predicate inventor as a by-product of full end-to-end training; however, further systematic investigation is required to confirm this effect.

A third direction involves building deeper neural DNF-based architectures. All neural DNF-based models developed in this thesis consist of only two trainable layers: a conjunctive layer and a disjunctive layer. Adding further trainable layers would increase the expressivity of the model. A conjunctive layer paired with a disjunctive layer can form a single neural DNF block, and such blocks can be stacked to form a multi-layer neural DNF-based model, as illustrated in Figure 8.1, thereby supporting *internal auxiliary* predicate invention. Internal auxiliary predicate invention differs from the *primary* predicate invention introduced in Section 7.4.1, which is employed in the tasks of Section 6.3.4 and Section 7.5 to convert unstructured inputs into structured predicates. Instead, the auxiliary predicates produced by each neural DNF block constitute a form of hierarchical knowledge learning: together with the task-input predicates, they are made accessible to all subsequent blocks, yielding a

hierarchical model structure similar to that of HRI [64]. The final neural DNF block then acts as the 'task-specific' rule learner, with its output activation set to either tanh or mutex-tanh depending on the task. The input predicates may originate either from the task itself or from predicates invented by other neural architectures.

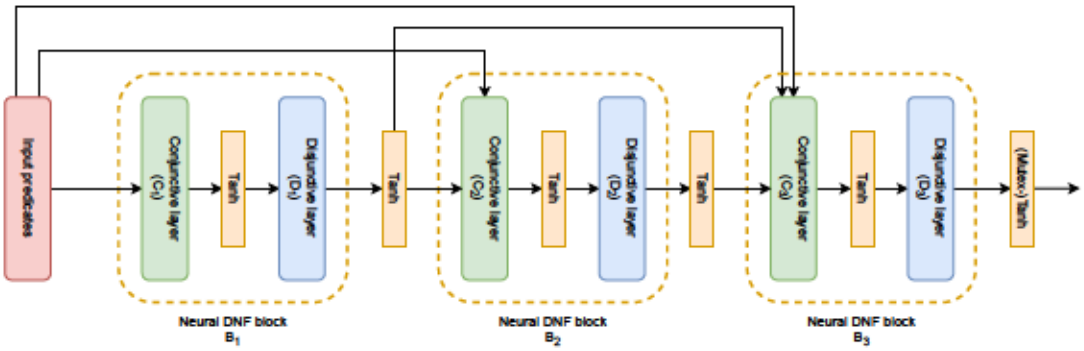


Figure 8.1: A proposal of a multi-layer neural DNF-based model with three neural DNF blocks. Block B_t can access all predicates from B_1 to B_{t-1}, as well as the input predicates.

A fourth direction is to investigate applying training tricks or developing new training regimes. Both the disentanglement approach and the harder logical constraint with the modified bias suggest that conventional gradient-based training might not be the optimal way to train a neural DNF-based model with good performance while maintaining a faithful logical interpretation. Further research into training regimes may help optimise the trade-off between performance and interpretability for neural DNF-based models. Below are some potential ways to improve the training process.

Constrained optimisation. One potential improvement is to incorporate constrained optimisation techniques in training. Works such as [112, 89] have utilised the Lagrangian function as their central mathematical tool for handling complex constraints. The Lagrangian combines the objective and all constraints into a single function using dual variables, defined as:

$$L(\theta, \lambda) = f(\theta) + \lambda g(\theta)$$

where λ is called the Lagrangian multiplier, and the optimum is reached when $\partial L/\partial\theta = 0$ and $\partial L/\partial\lambda = 0$. The objective function (i.e. the loss function) $f(\theta)$ is subject to an equality constraint $g(\theta) = 0$. In the case of training a neural DNF-based model, $g(\theta)$ will be the weight requirement, where $g(w_{ij}) = (w_{ij} - 6)(6 - w_{ij})w_{ij}$ for each weight w_{ij} in the model. Other works such as [166, 158] use projected gradient descent, where parameters are first updated using standard gradient steps and then projected onto the constraint set. These constrained optimisation techniques may be more effective to encode the weight requirements of a neural DNF-based model as a 'harder' constraint instead of the current auxiliary loss function (i.e. $|w||6 - |w||$).

Dropouts. The thesis did not explore the widely used dropout technique. Using dropouts during training, especially in the disjunctive layers, may prevent disjunctive nodes from co-adapting and force the neural DNF-based model to learn more robust features that do not rely on the presence of specific conjuncts.

AdamW. While the thesis does not use L_2 regularisation, it remains unclear whether the auxiliary loss $|w||6 - |w||$ in the loss function exhibits similar ineffectiveness as L_2 in Adam [87]. AdamW [97] separates weight decay from the gradient-based update and should be investigated if an adaptation of AdamW can better enforce the weight penalty $|w||6 - |w||$.

Gradient-free learning. Gradient-free learning is another area of optimisation. Unlike gradient-based optimisation, it does not require the computation of gradients. It is more suitable for problems that are not differentiable or that have a non-convex loss surface, which might be the case for the neural DNF-based models. Genetic algorithms are evolutionary algorithms that are inspired by the process of natural selection [146]. At each generation, the algorithm creates a population of candidate solutions, and evaluates their fitness. The fittest solutions are selected for reproduction, and the offspring are created through crossover and mutation. The process repeats until the termination condition is met. Simulated Annealing (SA) algorithm [129] is another gradient-free learning algorithm inspired by the process of annealing in metallurgy, where a material is heated and then slowly cooled to increase its crystal size and reduce defects. SA starts with a high 'temperature' that allows acceptance of worse solutions to escape local optima, then gradually cools down to converge on a global optimum. Going from gradient descent to gradient-free learning is a significant change in the training process, which might bring new challenges and opportunities.

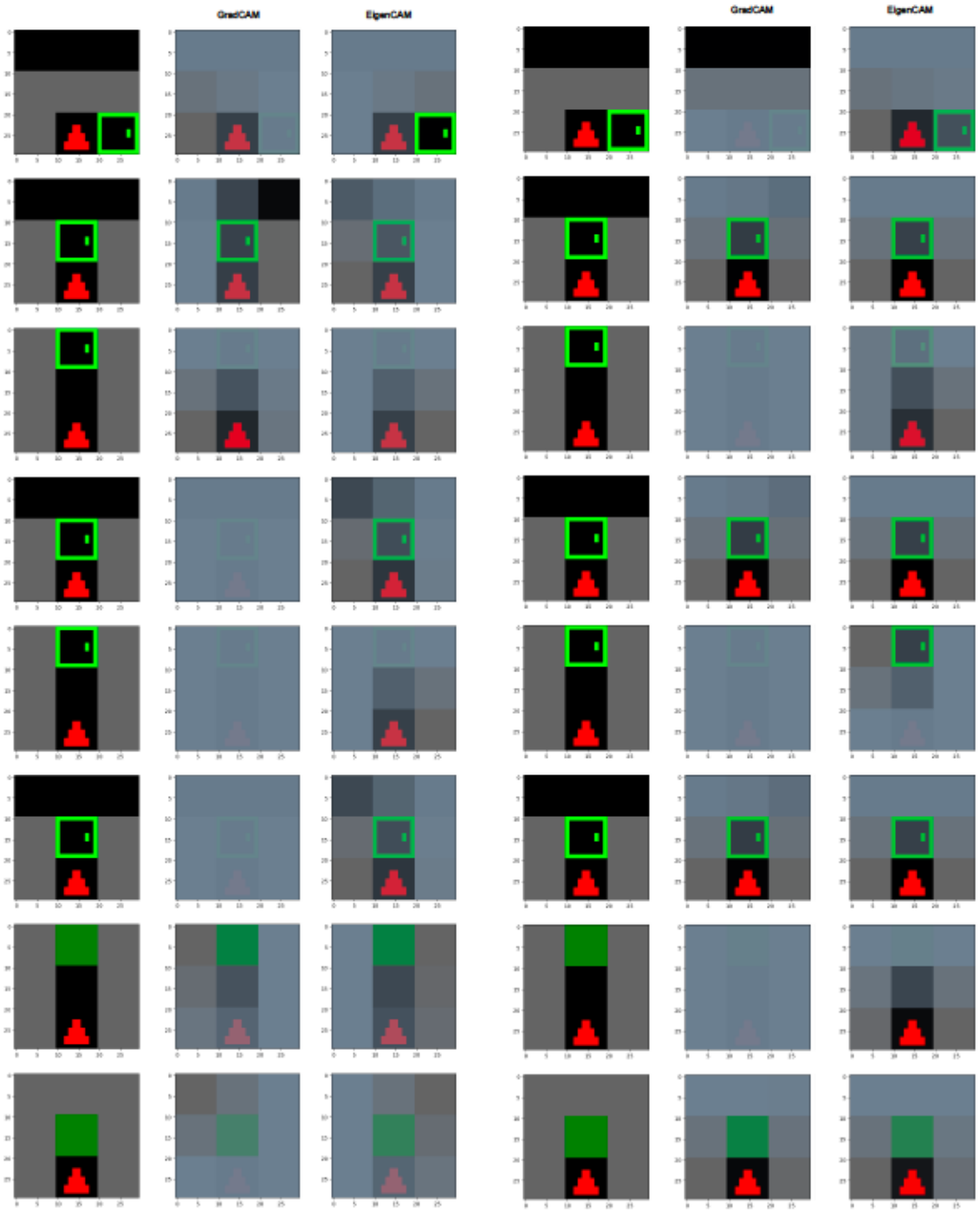
8.2.2 Applications in Real-life Domains

Aside from the applications in classification and reinforcement learning, the neural DNF-based models can also be applied in other real-life domains. One domain with great potential and a need for interpretability for safety reasons is healthcare. The neural DNF-based models can be used to learn diagnosis rules from a set of patients with different diagnoses. The binary classification task on the BCC dataset [122] in Section 7.5 shows the model's ability to diagnose breast cancer based on tabular data of patients' resistin, glucose, age, and BMI. Apart from tabular data, many healthcare applications involve image-based 2D/3D scans [79] or 'time-series' patient records [78]. The end-to-end differentiability of the neural DNF-based models allows integration with powerful neural components such as CNNs, RNNs and transformers for processing such unstructured data, and the neural DNF-based models can be used to learn the diagnosis rules from high-level features extracted.

Another domain requiring interpretability is anomaly detection in chemical processing. The work in [8] shows a prototype of using ILP approaches (ILASP [92] or FastLAS [91]) to learn rules to detect process anomalies from simulated and structured chemical process data. Again, the end-to-end differentiability of the neural DNF-based models can be leveraged to learn from real-valued time-series data, and the scalability and lack of reliance on background knowledge compared to traditional ILP approaches mean that little domain knowledge or human engineering is required to start training the model.

Finally, ensuring the correctness of LLM outputs has become a rising requirement for safety and reliability reasons, and various neuro-symbolic approaches have been proposed to provide logical constraints to LLMs. Ctrl-G [169] constrains an LLM with a distillable Hidden Markov Model (HMM) with logical constraints represented as deterministic finite automata (DFA). At inference

time, the HMM is conditioned on the DFA and the computed conditional probability is used to guide LLM generation. SEM-CTRL [6] proposes an approach using Answer Set Grammar (ASG) to encode syntactic and semantic constraints with token-level Monte Carlo Tree Search (MCTS) to ensure the correctness of LLM outputs. A subsequent study [5] builds on top of SEM-CTRL and learns context-sensitive constraints with ILASP [92], removing the need for manual engineering of the constraints. Using the bidirectional neural-bivalent-logic translation property, the neural DNF-based models can also be used to encode or learn logical constraints at the decoding process, and the end-to-end differentiability allows potential integration and parallel inference, which the current approaches lack.



(a) CAMs of the CNN encoder connected to an MLP actor.

(b) CAMs of the CNN encoder connected to a neural DNF-MT actor.

Figure 8.2: Grad-CAMs [139] and EigenCAMs [111] of the CNNs connected to an MLP (8.2a) and neural DNF-MT (8.2b) actor, at each unique step in the Door Corridor environment from Section 6.3.4. For each sub-figure, the left-to-right columns show the original inputs to the CNN, the Grad-CAMs, and the EigenCAMs in order. For each cell in the CAMs, the less greyed-out it is, the higher the activation it represents.

Bibliography

[1] Eu ai act, 2024.

[2] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9525–9536, Red Hook, NY, USA, 2018. Curran Associates Inc.

[3] Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck, and Antonio Vergari. Semantic probabilistic layers for neuro-symbolic learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[4] Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022)*, volume 180 of *Proceedings of Machine Learning Research*, pages 43–53. PMLR, 2022.

[5] Mohammad Albinhassan, Pranava Madhyastha, Mark Law, and Alessandra Russo. Learning and enforcing context-sensitive control for LLMs. In Jin Zhao, Mingyang Wang, and Zhu Liu, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 834–842, Vienna, Austria, July 2025. Association for Computational Linguistics.

[6] Mohammad Albinhassan, Pranava Madhyastha, and Alessandra Russo. `SEM-CTRL`: Semantically controlled decoding, 2025.

[7] Bradley P. Allen, Prateek Chhikara, Thomas Macaulay Ferguson, Filip Ilievski, and Paul Groth. Sound and complete neurosymbolic reasoning with llm-grounded interpretations. In Leilani H. Gilpin, Eleonora Giunchiglia, Pascal Hitzler, and Emile van Krieken, editors, *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*, volume 284 of *Proceedings of Machine Learning Research*, pages 392–419. PMLR, 08–10 Sep 2025.

[8] Julien Amblard, Alessandra Russo, and Gürkan Sin. On neuro-symbolic ai for abnormal event detection in process safety. *Chemical Engineering Transactions*, 116:445–450, Jun. 2025.

[9] Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. In *Proceedings of the 34th International*

Conference on Neural Information Processing Systems, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[10] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021.

[11] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, April 2024.

[12] Leo Ardon, Daniel Furelos-Blanco, Roko Parac, and Alessandra Russo. Form: Learning expressive and transferable first-order logic reward machines. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '25, pages 142–151, Richland, SC, 2025. International Foundation for Autonomous Agents and Multiagent Systems.

[13] Yaniv Aspis, Mohammad Albinhassan, Jorge Lobo, and Alessandra Russo. Embed2rule scalable neuro-symbolic learning via latent space weak-labelling. In Tarek R. Besold, Artur d'Avila Garcez, Ernesto Jimenez-Ruiz, Roberto Confalonieri, Pranava Madhyastha, and Benedikt Wagner, editors, *Neural-Symbolic Learning and Reasoning*, pages 195–218, Cham, 2024. Springer Nature Switzerland.

[14] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *CoRR*, abs/2204.05862, 2022.

[15] Kexin Gu Baugh, Nuri Cingillioglu, and Alessandra Russo. Neuro-symbolic rule learning in real-world classification tasks. In Andreas Martin, Hans-Georg Fill, Aurona Gerber, Knut Hinkelmann, Doug Lenat, Reinhard Stolle, and Frank van Harmelen, editors, *Proceedings of the AAAI 2023 Spring Symposium on Challenges Requiring the Combination of Machine Learning and Knowledge Engineering (AAAI-MAKE 2023)*, volume Vol-3433. CEUR Workshop Proceedings, 2023.

[16] Kexin Gu Baugh, Luke Dickens, and Alessandra Russo. Neural dnf-mt: A neuro-symbolic approach for learning interpretable and editable policies. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '25, pages 252–260, Richland, SC, 2025. International Foundation for Autonomous Agents and Multiagent Systems.

[17] Kexin Gu Baugh, Vincent Perreault, Matthew Baugh, Luke Dickens, Katsumi Inoue, and Alessandra Russo. Disentangling neural disjunctive normal form models. In Leilani H. Gilpin, Eleonora Giunchiglia, Pascal Hitzler, and Emile van Krieken, editors, *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*, volume 284 of *Proceedings of Machine Learning Research*, pages 463–493. PMLR, 08–10 Sep 2025.

[18] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[19] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[20] Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, 36(21):12809–12844, 2024.

[21] Jock Blackard. Covertype. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50K5N.

[22] Marko Bohanec. Car Evaluation. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C5JP48.

[23] Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *CoRR*, abs/1809.02193, 2018.

[24] Yushi Cao, Zhiming Li, Tianpei Yang, Hao Zhang, Yan Zheng, Yi Li, Jianye Hao, and Yang Liu. Galois: boosting deep reinforcement learning via generalizable logic synthesis. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[25] Álvaro Chaos, Max Aldana, Carlos Espinosa-Soto, Berenice García Ponce de León, Adriana Garay Arroyo, and Elena R. Alvarez-Buylla. From genes to flower patterns and evolution: Dynamic models of gene regulatory networks. *Journal of Plant Growth Regulation*, 25(4):278–289, Dec 2006.

[26] Theo Charalambous, Yaniv Aspis, and Alessandra Russo. Neuralfastlas: Fast logic-based learning from raw data, 2023.

[27] Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

[28] Nuri Cingillioglu and Alessandra Russo. pix2rule: End-to-end neuro-symbolic rule learning. In Artur d'Avila Garcez and Ernesto Jiménez-Ruiz, editors, *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy 2021) as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021)*. CEUR Workshop Proceedings, 2021.

[29] Andrew Cropper, Sebastijan Dumančić, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Machine Learning*, 111(1):147–172, Jan 2022.

[30] Andrew Cropper and Stephen H. Muggleton. Metagol system. https://github.com/metagol/metagol, 2016.

[31] Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. Ffnsl: Feed-forward neural-symbolic learner. *Machine Learning*, 112(2):515–569, Feb 2023.

[32] Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. Neuro-symbolic learning of answer set programs from raw data. In Edith Elkind, editor, *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 3586–3596. International Joint Conferences on Artificial Intelligence Organization, 8 2023. Main Track.

[33] Wang-Zhou Dai and Stephen Muggleton. Abductive knowledge induction from raw data. In Zhi-Hua 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.

[34] Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. In Edith 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.

[35] Maria I. Davidich and Stefan Bornholdt. Boolean network model predicts cell cycle sequence of fission yeast. *PLOS ONE*, 3(2):1–8, 02 2008.

[36] Artur S. d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *CoRR*, abs/2012.05876, 2020.

[37] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: a probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[38] Oliver Deane and Oliver Ray. Neuro-symbolic inverse constrained reinforcement learning. In Leilani H. Gilpin, Eleonora Giunchiglia, Pascal Hitzler, and Emile van Krieken, editors,

Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning, volume 284 of *Proceedings of Machine Learning Research*, pages 913–925. PMLR, 08–10 Sep 2025.

[39] Quentin Delfosse, Hikaru Shindo, Devendra Dhami, and Kristian Kersting. Interpretable and explainable logical policies via neurally guided symbolic abstraction. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 50838–50858. Curran Associates, Inc., 2023.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[41] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[42] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 2327–2333. AAAI Press, 2015.

[43] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[44] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1):7–52, Apr 2001.

[45] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[46] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[47] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*, 2020.

[48] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, Feb 2017.

[49] Richard Evans, Matko Bošnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek Sergot. Making sense of raw input (extended abstract). In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5727–5731. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Journal Track.

[50] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[51] Richard Evans, José Hernández-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek J. Sergot. Making sense of sensory input. *CoRR*, abs/1910.02227, 2019.

[52] Adrien Fauré, Aurélien Naldi, Claudine Chaouiya, and Denis Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 07 2006.

[53] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. DL2: Training and querying neural networks with logic. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1931–1941. PMLR, 09–15 Jun 2019.

[54] Ricardo Fitas. Neuro-symbolic ai for advanced signal and image processing: A review of recent trends and future directions. *IEEE Access*, 13:143360–143376, 2025.

[55] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Hierarchies of reward machines. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, volume 202 of *Proceedings of Machine Learning Research*, pages 10494–10541. PMLR, 2023.

[56] Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. A differentiable first-order rule learner for inductive logic programming. *Artificial Intelligence*, 331:104108, 2024.

[57] Kun Gao, Hanpin Wang, Yongzhi Cao, and Katsumi Inoue. Learning from interpretation transition using differentiable logic programming semantics. *Machine Learning*, 111(1):123–145, Jan 2022.

[58] Leo Gao, Tom Dupre la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders. In *The Thirteenth International Conference on Learning Representations*, 2025.

[59] Xiaojie Gao, Yueming Jin, Qi Dou, and Pheng-Ann Heng. Automatic gesture recognition in robot-assisted surgery with reinforcement learning and tree search. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8440–8446, 2020.

[60] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski, Bowen, and Kenneth, editors, *Proceedings of International Logic Programming Conference and Symposium*, pages 1070–1080. MIT Press, 1988.

[61] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3681–3688, Jul. 2019.

[62] Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances*

in Neural Information Processing Systems, volume 33, pages 9662–9673. Curran Associates, Inc., 2020.

[63] Eleonora Giunchiglia, Alex Tatomir, Mihaela Cătălina Stoian, and Thomas Lukasiewicz. Ccn+: A neuro-symbolic framework for deep learning with requirements. *International Journal of Approximate Reasoning*, 171:109124, 2024. Synergies between Machine Learning and Reasoning.

[64] Claire Glanois, Zhaohui Jiang, Xuening Feng, Paul Weng, Matthieu Zimmer, Dong Li, Wulong Liu, and Jianye Hao. Neuro-symbolic hierarchical rule induction. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7583–7615. PMLR, 17–23 Jul 2022.

[65] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[66] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[67] Audobon Society Field Guide. Mushroom. UCI Machine Learning Repository, 1981. DOI: https://doi.org/10.24432/C5959T.

[68] M.M. Gupta and J. Qi. Theory of t-norms and fuzzy inference methods. *Fuzzy Sets and Systems*, 40(3):431–450, 1991. Fuzzy Logic and Uncertainty Modelling.

[69] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[70] Jianxing He, Sally L. Baxter, Jie Xu, Jiming Xu, Xingtao Zhou, and Kang Zhang. The practical implementation of artificial intelligence technologies in medicine. *Nature Medicine*, 25(1):30–36, Jan 2019.

[71] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 7265–7273. Curran Associates, Inc., 2019.

[72] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[73] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116. PMLR, 10–15 Jul 2018.

[74] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning*, 94(1):51–79, Jan 2014.

[75] Adam Ishay, Zhun Yang, and Joohyung Lee. Leveraging large language models to generate answer set programs. In *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, KR '23, 2023.

[76] Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-kast: top-k always sparse training. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[77] Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3110–3119. PMLR, 09–15 Jun 2019.

[78] Alistair Johnson, Lucas Bulgarelli, Tom Pollard, Brian Gow, Benjamin Moody, Steven Horng, Leo Anthony Celi, and Roger Mark. MIMIC-IV, 2024.

[79] Alistair Johnson, Tom Pollard, Roger Mark, Seth Berkowitz, and Steven Horng. MIMIC-CXR Database, 2024.

[80] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998.

[81] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[82] Stuart Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224(5215):177–178, Oct 1969.

[83] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *CoRR*, abs/1807.00412, 2018.

[84] Daiki Kimura, Masaki Ono, Subhajit Chaudhury, Ryosuke Kohita, Akifumi Wachi, Don Joven Agravante, Michiaki Tatsubori, Asim Munawar, and Alexander Gray. Neuro-symbolic reinforcement learning with first-order logic. In Marie-Francine Moens, Xuanjing Huang, Lucia

Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3505–3511, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[85] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, Dumitru Erhan, and Been Kim. *The (Un)reliability of Saliency Methods*, pages 267–280. Springer International Publishing, Cham, 2019.

[86] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

[87] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[88] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *ICML*, pages 5338–5348, 2020.

[89] James Kotary and Ferdinando Fioretto. Learning constrained optimization with deep augmented lagrangian methods, 2024.

[90] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[91] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2877–2885, Apr. 2020.

[92] Mark Law, Alessandra Russo, and Krysia Broda. The complexity and generality of learning answer set programs. *Artificial Intelligence*, 259:110–146, 2018.

[93] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[94] Fangting Li, Tao Long, Ying Lu, Qi Ouyang, and Chao Tang. The yeast cell-cycle network is robustly designed. *Proceedings of the National Academy of Sciences*, 101(14):4781–4786, 2004.

[95] Vladimir Lifschitz. *Answer set programming*. Springer Nature, Cham, Switzerland, 2019.

[96] Zhong Qiu Lin, Mohammad Javad Shafiee, Stanislav Bochkarev, Michael St. Jules, Xiaoyu Wang, and Alexander Wong. Do explanations reflect decisions? A machine-centric strategy to quantify the performance of explainability algorithms. *CoRR*, abs/1910.07387, 2019.

[97] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[98] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[99] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30:4765–4774, 2017.

[100] Alireza Makhzani and Brendan Frey. k-sparse autoencoders. 2014.

[101] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[102] Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate Inference for Neural Probabilistic Logic Programming. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, pages 475–486, 11 2021.

[103] Andrei Margeloiu, Matthew Ashman, Umang Bhatt, Yanzhi Chen, Mateja Jamnik, and Adrian Weller. Do concept bottleneck models learn as intended? *CoRR*, abs/2105.04289, 2021.

[104] Scott Mayer McKinney, Marcin Sieniek, Varun Godbole, Jonathan Godwin, Natasha Antropova, Hutan Ashrafian, Trevor Back, Mary Chesus, Greg S. Corrado, Ara Darzi, Mozziyar Etemadi, Florencia Garcia-Vicente, Fiona J. Gilbert, Mark Halling-Brown, Demis Hassabis, Sunny Jansen, Alan Karthikesalingam, Christopher J. Kelly, Dominic King, Joseph R. Ledsam, David Melnick, Hormuz Mostofi, Lily Peng, Joshua Jay Reicher, Bernardino Romera-Paredes, Richard Sidebottom, Mustafa Suleyman, Daniel Tse, Kenneth C. Young, Jeffrey De Fauw, and Shravya Shetty. International evaluation of an ai system for breast cancer screening. *Nature*, 577(7788):89–94, Jan 2020.

[105] Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2020)*, pages 5182–5190, 2020.

[106] Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, and Sebastian Riedel. Towards neural theorem proving at scale, 2018.

[107] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[108] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[109] Stephen Muggleton. Inverse entailment and progol. *New Generation Computing*, 13(3):245–286, Dec 1995.

[110] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994. Special Issue: Ten Years of Logic Programming.

[111] Mohammed Bany Muhammad and Mohammed Yeasin. Eigen-cam: Class activation map using principal components. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE, 2020.

[112] Yatin Nandwani, Abhishek Pathak, Mausam, and Parag Singla. A primal dual formulation for deep learning with constraints. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[113] Marcio Nicolau, Anderson R. Tavares, Zhiwei Zhang, Pedro H. C. Avelar, João Marcos Flach, Luis DC Lamb, and Moshe Vardi. Understanding boolean function learnability on deep neural networks: Pac learning meets neurosymbolic models. In Leilani H. Gilpin, Eleonora Giunchiglia, Pascal Hitzler, and Emile van Krieken, editors, *Proceedings of The 19th International Conference on Neurosymbolic Learning and Reasoning*, volume 284 of *Proceedings of Machine Learning Research*, pages 719–735. PMLR, 08–10 Sep 2025.

[114] Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum, and Roger Levy. LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5153–5176, Singapore, December 2023. Association for Computational Linguistics.

[115] OpenAI. Openai baselines: Acktr & a2c. `https://openai.com/index/openai-baselines-acktr-a2c/`, Aug 2017.

[116] OpenAI. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`, 2025. Accessed: 2025-10-28.

[117] OpenAI. Sora 2 is here. `https://openai.com/index/sora-2/`, September 2025. Video and audio generation model release.

[118] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[119] Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore, December 2023. Association for Computational Linguistics.

[120] Mohit Pandey, Michael Fernandez, Francesco Gentile, Olexandr Isayev, Alexander Tropsha, Abraham C. Stern, and Artem Cherkasov. The transformational role of gpu computing and deep learning in drug discovery. *Nature Machine Intelligence*, 4(3):211–221, Mar 2022.

[121] Roko Parać, Lorenzo Nodari, Leo Ardon, Daniel Furelos-Blanco, Federico Cerutti, and Alessandra Russo. Learning robust reward machines from noisy labels. In *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning*, KR '24, 2024.

[122] Miguel Patrício, José Pereira, Joana Crisóstomo, Paulo Matafome, Manuel Gomes, Raquel Seiça, and Francisco Caramelo. Using resistin, glucose, age and bmi to predict the presence of breast cancer. *BMC Cancer*, 18(1):29, Jan 2018.

[123] Ali Payani and Faramarz Fekri. Inductive logic programming via differentiable deep neural logic networks. *CoRR*, abs/1906.03523, 2019.

[124] Vincent Perreault, Katsumi Inoue, Richard Labib, and Alain Hertz. Neural logic networks for interpretable classification, 2025.

[125] Yin Jun Phua and Katsumi Inoue. Variable assignment invariant neural networks for learning logic programs. In Tarek R. Besold, Artur d'Avila Garcez, Ernesto Jimenez-Ruiz, Roberto Confalonieri, Pranava Madhyastha, and Benedikt Wagner, editors, *Neural-Symbolic Learning and Reasoning*, pages 47–61, Cham, 2024. Springer Nature Switzerland.

[126] Potassco, the Potsdam Answer Set Solving Collection. Clingo: A grounder and solver for logic programs., 2022.

[127] Martin L. Puterman. Markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier, 1990.

[128] Luc de Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4943–4950. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Survey track.

[129] L.M. Rasdi Rere, Mohamad Ivan Fanany, and Aniati Murni Arymurthy. Simulated annealing algorithm for deep learning. *Procedia Computer Science*, 72:137–144, 2015. The Third Information Systems International Conference 2015.

[130] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[131] Ryan Riegel, Alexander G. Gray, Francois P. S. Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, Shajith Ikbal, Hima Karanam, Sumit Neelam, Ankita Likhyani, and Santosh K. Srivastava. Logical neural networks. *CoRR*, abs/2006.13155, 2020.

[132] Tim Rocktäschel and Sebastian Riedel. Learning knowledge base inference with neural theorem provers. In Jay Pujara, Tim Rocktaschel, Danqi Chen, and Sameer Singh, editors, *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 45–50, San Diego, CA, June 2016. Association for Computational Linguistics.

[133] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, May 2019.

[134] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends. *AI Communications*, 34(3):197–209, 2021.

[135] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[136] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[137] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[138] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[139] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[140] Prithviraj Sen, Breno WSR de Carvalho, Ryan Riegel, and Alexander Gray. Neuro-symbolic inductive logic programming with logical neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8212–8219, 2022.

[141] Jingyuan Sha, Hikaru Shindo, Kristian Kersting, and Devendra Singh Dhami. Neural-symbolic predicate invention: Learning relational concepts from visual scenes. In Artur d'Avila Garcez and Ernesto Jiménez-Ruiz, editors, *NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning*, pages 103–117. CEUR Workshop Proceedings, 2023.

[142] Hikaru Shindo, Masaaki Nishino, and Akihiro Yamamoto. Differentiable inductive logic programming for structured examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5034–5041, 2021.

[143] Hikaru Shindo, Viktor Pfanschilling, Devendra Singh Dhami, and Kristian Kersting. Alpha ilp: thinking visual scenes as differentiable logic programs. *Machine Learning*, 112(5):1465–1497, May 2023.

[144] Hikaru Shindo, Viktor Pfanschilling, Devendra Singh Dhami, and Kristian Kersting. Learning differentiable logic programs for abstract visual reasoning. *Machine Learning*, 113(11):8533–8584, Dec 2024.

[145] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017.

[146] S. N. Sivanandam and S. N. Deepa. *Introduction to genetic algorithms*. Springer, 2010.

[147] Ashwin Srinivasan. *The ALEPH Manual*, 2001.

[148] A.N. Srivastava and B. Zane-Ulman. Discovering recurring anomalies in text reports regarding complex space systems. In *2005 IEEE Aerospace Conference*, pages 3853–3862, 2005.

[149] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era . In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 843–852, Los Alamitos, CA, USA, October 2017. IEEE Computer Society.

[150] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[151] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[152] Alex Teboul. Diabetes health indicators dataset. Kaggle, 2021.

[153] Neil C. Thompson, Kristjan H. Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning. *CoRR*, abs/2007.05558, 2020.

[154] Abhinav Verma, Hoang M. Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[155] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5045–5054. PMLR, 10–15 Jul 2018.

[156] Celeste Veronese, Daniele Meli, and Alessandro Farinelli. Online inductive learning from answer sets for efficient reinforcement learning exploration. In Pierangela Bruno, Francesco Calimeri, Francesco Cauteruccio, and Giorgio Terracina, editors, *Hybrid Models for Coupling Deductive and Inductive Reasoning*, pages 93–106, Cham, 2025. Springer Nature Switzerland.

[157] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge J. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical report, 2011.

[158] Zifan Wang, Changxin Liu, Thomas Parisini, Michael M. Zavlanos, and Karl H. Johansson. Projected gradient descent for constrained decision-dependent optimization, 2025.

[159] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[160] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992.

[161] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. Chapter 10 - deep learning. In Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal, editors, *Data Mining (Fourth Edition)*, pages 417–466. Morgan Kaufmann, fourth edition edition, 2017.

[162] J. Wnek. MONK's Problems. UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5R30R.

[163] XiaoDan Wu, RuiChang Li, Zhen He, TianZhi Yu, and ChangQing Cheng. A value-based deep reinforcement learning model with human expertise in optimal treatment of sepsis. *npj Digital Medicine*, 6(1):15, Feb 2023.

[164] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5502–5511. PMLR, 10–15 Jul 2018.

[165] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In Christian 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. Main track.

[166] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. In *Computer Vision - ECCV 2020: 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, pages 635–652, Berlin, Heidelberg, 2020. Springer-Verlag.

[167] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew M. Botvinick, Oriol Vinyals, and Peter W. Battaglia. Relational deep reinforcement learning. *CoRR*, abs/1806.01830, 2018.

[168] Hanwei Zhang, Felipe Torres Figueroa, and Holger Hermanns. Saliency Maps Give a False Sense of Explanability to Image Classifiers: An empirical evaluation across methods and metrics. In Vu Nguyen and Hsuan-Tien Lin, editors, *Proceedings of the 16th Asian Conference on Machine Learning*, volume 260 of *Proceedings of Machine Learning Research*, pages 479–494. PMLR, 05–08 Dec 2025.

[169] Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc.

[170] Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, Paul Weng, Jianye Hao, Dong Li, and Wulong Liu. Differentiable logic machines. *CoRR*, abs/2102.11529, 2021.

[171] K.J Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

Appendix A

Semi-Symbolic Node and Neural DNF Model: Supplementary Material

A.1 Properties of Discretised Semi-Symbolic Nodes

Proof of Proposition 4.1.2. Because the node is discretised, all elements in the relevant set have an absolute value of 6, that is, $\forall i \in \mathcal{J}_{\mathbf{w}_d}.|w_{di}| = 6$. From Equation (4.2), the bias and the raw output of this discretised disjunctive node can be calculated as:

$$\beta = \sum_{i=1}^{N} |w_{di}| - \max_{i=1,\ldots,N} |w_{di}| = \sum_{i \in \mathcal{J}_{\mathbf{w}_d}} 6 - 6 = 6|\mathcal{J}_{\mathbf{w}_d}| - 6$$

$$f_{\mathbf{w}_d}(\mathbf{x}) = \sum_{i \in \mathcal{J}_{\mathbf{w}_d}} w_{di} x_i + 6|\mathcal{J}_{\mathbf{w}_d}| - 6$$

Since $\forall i \in \mathcal{J}_{\mathbf{w}_d}.\, [\text{sign}(w_{di}) \neq 0, |w_{di}| = 6, |x_i| = 1]$, this gives:

$$\forall i \in \mathcal{J}_{\mathbf{w}_d}.w_{di}x_i = \begin{cases} 6 & \text{if sign}(w_{di}) = x_i \\ -6 & \text{otherwise} \end{cases}$$

Thus, the raw output can be reformulated using the sign-matching and sign-mismatching sets:

$$\begin{aligned}
f_{\mathbf{w}_d}(\mathbf{x}) &= \sum_{i \in \mathcal{J}_{\mathbf{w}_d}} w_{di} x_i + 6|\mathcal{J}_{\mathbf{w}_d}| - 6 \\
&= \sum_{i \in \mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}} 6 + \sum_{i \in \mathcal{J}^-_{\mathbf{w}_d \odot \mathbf{x}}} (-6) + 6|\mathcal{J}_{\mathbf{w}_d}| - 6 \\
&= 6 \left(\sum_{i \in \mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^-_{\mathbf{w}_d \odot \mathbf{x}}} 1 + |\mathcal{J}_{\mathbf{w}_d}| - 1 \right) \\
&= 6 \left(|\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| - |\mathcal{J}^-_{\mathbf{w}_d \odot \mathbf{x}}| + |\mathcal{J}_{\mathbf{w}_d \odot \mathbf{x}}| - 1 \right) \\
&= 6 \left(2|\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| - 1 \right) \qquad \text{given } |\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| + |\mathcal{J}^-_{\mathbf{w}_d \odot \mathbf{x}}| = |\mathcal{J}_{\mathbf{w}_d}|
\end{aligned}$$

Since $|\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| \in \mathbb{N}$, the lower bound of $2|\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| - 1$ is -1, and $(2|\mathcal{J}^+_{\mathbf{w}_d \odot \mathbf{x}}| - 1) \in \{-1, 1, 3, 5, \ldots\}$. Thus, $f_{\mathbf{w}_d}(\mathbf{x}) \in \{-6, 6, 18, 30, \ldots\}$. ■

A.2 Proofs of Neural-Bivalent-Logic Translation

Proof of Theorem 4.2.1. The bias of the discretised conjunctive node is calculated as follows, using Equation (4.2):

$$\beta_c = \max_{i=1,\ldots,N} |w_{c,i}| - \sum_{i=1}^{N} |w_{c,i}| = 6 - \sum_{i \in \mathcal{J}_{\mathbf{w}_c}} 6 = 6 - 6|\mathcal{J}_{\mathbf{w}_c}|$$

The output of the conjunctive node is calculated as follows:

$$\begin{aligned}
f_{\mathbf{w}_c}(\mathbf{x}) &= \sum_{i=1}^{N} w_{c,i} x_i + \beta_c \\
&= 6 \sum_{i \in \mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}} x_i - 6 \sum_{i \in \mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}} x_i + (6 - 6|\mathcal{J}_{\mathbf{w}_c}|)
\end{aligned}$$

where $\mathcal{J}^+_{\mathbf{w}_c \odot \mathbf{x}}$ and $\mathcal{J}^-_{\mathbf{w}_c \odot \mathbf{x}}$ are the sign-matching and sign-mismatching sets.

The relevant set $\mathcal{J}_{\mathbf{w}_c}$ can be further split into four mutually exclusive subsets, using the intersection of sign-based index sets:

$$\begin{aligned}
\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_c}, \text{sign}(w_{c,i}) = 1, \text{sign}(x_i) = 1\} \\
\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_c}, \text{sign}(w_{c,i}) = 1, \text{sign}(x_i) = -1\} \\
\mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_c}, \text{sign}(w_{c,i}) = -1, \text{sign}(x_i) = -1\} \\
\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_c}, \text{sign}(w_{c,i}) = -1, \text{sign}(x_i) = 1\}
\end{aligned}$$

And by construction, the following equalities hold:

$$\begin{aligned}
\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \cup \mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}} &= \mathcal{J}^{+}_{\mathbf{w}_c \odot \mathbf{x}} && \text{Sign-matching between } \mathbf{w}_c \text{ and } \mathbf{x} \\
\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} \cup \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_c \odot \mathbf{x}} && \text{Sign-mismatching between } \mathbf{w}_c \text{ and } \mathbf{x} \\
\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \cup \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} &= \mathcal{J}^{+}_{\mathbf{w}_c} && \text{Positive-sign relevant set} \\
\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}} \cup \mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_c} && \text{Negative-sign relevant set}
\end{aligned}$$

Thus, the raw output of the conjunctive node can be calculated as:

$$\begin{aligned}
f_{\mathbf{w}_c}(\mathbf{x}) &= 6 \sum_{i \in \mathcal{J}^{+}_{\mathbf{w}_c \odot \mathbf{x}}} x_i - 6 \sum_{i \in \mathcal{J}^{-}_{\mathbf{w}_c \odot \mathbf{x}}} x_i + (6 - 6|\mathcal{J}_{\mathbf{w}_c}|) \\
&= 6 \left(\sum_{i \in \mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}}} 1 + \sum_{i \in \mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}} 1 + 1 - |\mathcal{J}_{\mathbf{w}_c}| \right) \\
&= 6 \left(|\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}}| + |\mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}}| - |\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| - |\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - |\mathcal{J}_{\mathbf{w}_c}| + 1 \right) \\
&= 6 \left(1 - 2|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - 2|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| \right) \qquad \text{(A.1)}
\end{aligned}$$

Based on Proposition 4.1.1, the two cases of $f_{\mathbf{w}_c}(\mathbf{x})$ are considered.

Case 1: $f_{\mathbf{w}_c}(\mathbf{x}) = 6 > 0$. The conjunctive node's bivalent logical interpretation $b_c = \top$. It must be proven that c from the translation defined in Equation (4.20) is in the answer set (i.e. c is true).

By $f_{\mathbf{w}_c}(\mathbf{x}) = 6$ and Equation (A.1), the following equalities hold:

$$\begin{aligned}
1 - 2|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - 2|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| &= 1 \\
|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| + |\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| &= 0 \\
\Rightarrow \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} = \emptyset \wedge \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}} &= \emptyset \\
\Rightarrow \mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} = \mathcal{J}^{+}_{\mathbf{w}_c} \wedge \mathcal{J}^{-,-}_{\mathbf{w}_c,\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_c}
\end{aligned}$$

Using these equalities and the translation defined in Equation (4.21), the set of ASP facts translated from input $\mathbf{x}$ is:

$$\begin{aligned}
\mathcal{P}_{\mathbf{x},\text{facts}} &= \{\mathrm{a}_i \mid i \in \mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \cup i \in \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}\} \\
&= \{\mathrm{a}_i \mid i \in \mathcal{J}^{+}_{\mathbf{w}_c}\}
\end{aligned}$$

This gives the following implications: the conjunction $\bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_c}} \mathrm{a}_i$ is true because all the literals a_i where $i \in \mathcal{J}^{+}_{\mathbf{w}_c}$ are present as facts in $\mathcal{P}_{\mathbf{x},\text{facts}}$; and the conjunction $\bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_c}} (\texttt{not } \mathrm{a}_i)$ is also true because none of the literals $\texttt{not } a_i$ where $i \in \mathcal{J}^{-}_{\mathbf{w}_c}$ is present as a fact in $\mathcal{P}_{\mathbf{x},\text{facts}}$. Thus, the rule body $\bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_c}} \mathrm{a}_i, \bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_c}} (\texttt{not } \mathrm{a}_i)$ is true, and the rule head c is present in the answer set (i.e. c is true) in the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and the rule from the translation defined in Equation (4.20).

Case 2: $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6 < 0$. The conjunctive node's bivalent logical interpretation $b_c = \bot$. It

must be proven that c is not in the answer set (i.e. c is false).

By $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6$ and Equation (A.1):

$$\begin{aligned} 1 - 2|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - 2|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| &\leq -1 \\ |\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| + |\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| &\geq 1 \end{aligned} \tag{A.2}$$

Recall the set of translated ASP facts from input $\mathbf{x}$ is $\mathcal{P}_{\mathbf{x},\text{facts}}$ as defined in Equation (4.21):

$$\mathcal{P}_{\mathbf{x},\text{facts}} = \{\mathrm{a}_i \mid i \in \mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \cup i \in \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}\}$$

The inequality can be split into two cases.

(2.a): If $|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| \geq 1$. By construction, if $i \in \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}$, then $i \in \mathcal{J}^{-}_{\mathbf{w}_c}$. Therefore, regardless of the value of $|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}|$, there exists at least one literal a_i where $i \in \mathcal{J}^{-}_{\mathbf{w}_c}$ that is present as a fact in $\mathcal{P}_{\mathbf{x},\text{facts}}$, causing the `not` a_i to be false and the conjunction $\bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_c}} (\texttt{not}\ \mathrm{a}_i)$ to be false. Thus, the body of the rule from Equation (4.20) is false, and the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and this rule does not have c in its answer set (i.e. c is false).

(2.b): If $|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| \geq 1$. Since $\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \cup \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} = \mathcal{J}^{+}_{\mathbf{w}_c}$, there is a strict subset relationship $\mathcal{J}^{+,+}_{\mathbf{w}_c,\mathbf{x}} \subset \mathcal{J}^{+}_{\mathbf{w}_c}$. Therefore, there exists at least one literal a_i where $i \in \mathcal{J}^{+}_{\mathbf{w}_c}$ that is not present as a fact in $\mathcal{P}_{\mathbf{x},\text{facts}}$, causing the conjunction $\bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_c}} \mathrm{a}_i$ to be false. Thus, the body of the rule from Equation (4.20) is false, and the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and this rule does not have c in its answer set (i.e. c is false).

Together, these two cases establish that the bivalent logical interpretation b_c of the conjunctive semi-symbolic node has the same truth value as the rule head c in the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and the rule $c \leftarrow \bigwedge_{i \in \mathcal{J}^{+}_{\mathbf{w}_c}} \mathrm{a}_i, \bigwedge_{i \in \mathcal{J}^{-}_{\mathbf{w}_c}} (\texttt{not}\ \mathrm{a}_i)$ from Equation (4.20), whether $f_{\mathbf{w}_c}(\mathbf{x}) = 6$ or $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6$. ■

Proof of Theorem 4.2.2. The bias of the discretised disjunctive node is calculated as follows, using Equation (4.2):

$$\beta_d = \sum_{i=1}^{N} |w_{d,i}| - \max_{i=1,\ldots,N} |w_{d,i}| = \sum_{i \in \mathcal{J}_{\mathbf{w}_d}} 6 - 6 = 6|\mathcal{J}_{\mathbf{w}_d}| - 6$$

The output of the disjunctive node is calculated as follows:

$$\begin{aligned} f_{\mathbf{w}_d}(\mathbf{x}) &= \sum_{i=1}^{N} w_{d,i} x_i + \beta_d \\ &= 6 \sum_{i \in \mathcal{J}^{+}_{\mathbf{w}_d \odot \mathbf{x}}} x_i - 6 \sum_{i \in \mathcal{J}^{-}_{\mathbf{w}_d \odot \mathbf{x}}} x_i + (6|\mathcal{J}_{\mathbf{w}_d}| - 6) \end{aligned}$$

where $\mathcal{J}^{+}_{\mathbf{w}_d \odot \mathbf{x}}$ and $\mathcal{J}^{-}_{\mathbf{w}_d \odot \mathbf{x}}$ are the sign-matching and sign-mismatching sets.

The relevant set $\mathcal{J}_{\mathbf{w}_d}$ can be further split into four mutually exclusive subsets, using the inter-

section of sign-based index sets:

$$\begin{aligned}
\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_\mathrm{d}}, \mathrm{sign}(w_{\mathrm{d},i}) = 1, \mathrm{sign}(x_i) = 1\} \\
\mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_\mathrm{d}}, \mathrm{sign}(w_{\mathrm{d},i}) = 1, \mathrm{sign}(x_i) = -1\} \\
\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_\mathrm{d}}, \mathrm{sign}(w_{\mathrm{d},i}) = -1, \mathrm{sign}(x_i) = -1\} \\
\mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \{i \mid i \in \mathcal{J}_{\mathbf{w}_\mathrm{d}}, \mathrm{sign}(w_{\mathrm{d},i}) = -1, \mathrm{sign}(x_i) = 1\}
\end{aligned}$$

And by construction, the following equalities hold:

$$\begin{aligned}
\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} \cup \mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d} \odot \mathbf{x}} && \text{Sign-matching between } \mathbf{w}_\mathrm{d} \text{ and } \mathbf{x} \\
\mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} \cup \mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_\mathrm{d} \odot \mathbf{x}} && \text{Sign-mismatching between } \mathbf{w}_\mathrm{d} \text{ and } \mathbf{x} \\
\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} \cup \mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d}} && \text{Positive-sign relevant set} \\
\mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} \cup \mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_\mathrm{d}} && \text{Negative-sign relevant set}
\end{aligned}$$

Thus, the raw output of the disjunctive node can be calculated as:

$$\begin{aligned}
f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) &= 6 \sum_{i \in \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d} \odot \mathbf{x}}} x_i - 6 \sum_{i \in \mathcal{J}^{-}_{\mathbf{w}_\mathrm{d} \odot \mathbf{x}}} x_i + (6|\mathcal{J}_{\mathbf{w}_\mathrm{d}}| - 6) \\
&= 6 \left(\sum_{i \in \mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}} 1 + \sum_{i \in \mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}} 1 - \sum_{i \in \mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}} 1 + |\mathcal{J}_{\mathbf{w}_\mathrm{d}}| - 1 \right) \\
&= 6 \left(|\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| + |\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| - |\mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| - |\mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| + |\mathcal{J}_{\mathbf{w}_\mathrm{d}}| - 1 \right) \\
&= 6 \left(2|\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| - 1 \right)
\end{aligned} \tag{A.3}$$

Based on Proposition 4.1.2, the two cases of $f_{\mathbf{w}_\mathrm{d}}(\mathbf{x})$ are considered.

Case 1: $f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) \geq 6 > 0$. The disjunctive node's bivalent logical interpretation $b_\mathrm{d} = \top$. It must be proven that d from the translation defined in Equation (4.24) is in the answer set (i.e. d is true).

By $f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) \geq 6$ and Equation (A.3), the following inequality holds:

$$\begin{aligned}
2|\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| - 1 &\geq 1 \\
|\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| + |\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| &\geq 1
\end{aligned} \tag{A.4}$$

Recall the set of translated ASP facts from input $\mathbf{x}$ is $\mathcal{P}_{\mathbf{x},\mathrm{facts}}$ as defined in Equation (4.25):

$$\mathcal{P}_{\mathbf{x},\mathrm{facts}} = \{\mathrm{a}_i \mid i \in \mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} \cup i \in \mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}\}$$

The inequality can be split into two cases.

(1.a): If $|\mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}| \geq 1$. By construction, if $i \in \mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}}$, then $i \in \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d}}$. Therefore, regardless of the value of $|\mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}}|$, there exists at least one literal a_i where $i \in \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d}}$ that is present as a fact in $\mathcal{P}_{\mathbf{x},\mathrm{facts}}$, causing the body of rule $\mathrm{d} \leftarrow \mathrm{a}_i$ to be true. Thus, the combined program of $\mathcal{P}_{\mathbf{x},\mathrm{facts}}$ and the rules

from Equation (4.24) has d in its answer set (i.e. d is true).

(1.b): If $|\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| \geq 1$. By construction, if $i \in \mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}$, then $i \in \mathcal{J}^{-}_{\mathbf{w}_d}$ and $x_i = -1$, meaning $\mathrm{a}_i \notin \mathcal{P}_{\mathbf{x},\text{facts}}$. Therefore, regardless of the value of $|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}|$, there exists at least one literal `not` a_i where $i \in \mathcal{J}^{-}_{\mathbf{w}_d}$ that is not present as a fact in $\mathcal{P}_{\mathbf{x},\text{facts}}$, causing the body of rule $\mathrm{d} \leftarrow$ `not` a_i to be true. Thus, the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and the rules from Equation (4.24) has d in its answer set (i.e. d is true).

Case 2: $f_{\mathbf{w}_d}(\mathbf{x}) = -6 < 0$. The disjunctive node's bivalent logical interpretation $b_d = \bot$. It must be proven that d is not in the answer set (i.e. d is false).

By $f_{\mathbf{w}_d}(\mathbf{x}) = -6$ and Equation (A.3):

$$\begin{aligned} 2|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| - 1 &= -1 \\ |\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + |\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| &= 0 \\ \Rightarrow \mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}} = \emptyset \wedge \mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}} &= \emptyset \\ \Rightarrow \mathcal{J}^{+,-}_{\mathbf{w}_d,\mathbf{x}} = \mathcal{J}^{+}_{\mathbf{w}_d} \wedge \mathcal{J}^{-,+}_{\mathbf{w}_d,\mathbf{x}} &= \mathcal{J}^{-}_{\mathbf{w}_d} \end{aligned}$$

Using these equalities and the translation defined in Equation (4.25), the set of ASP facts translated from input $\mathbf{x}$ is:

$$\begin{aligned} \mathcal{P}_{\mathbf{x},\text{facts}} &= \{\mathrm{a}_i \mid i \in \mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}} \cup i \in \mathcal{J}^{-,+}_{\mathbf{w}_d,\mathbf{x}}\} \\ &= \{\mathrm{a}_i \mid i \in \mathcal{J}^{-}_{\mathbf{w}_d}\} \end{aligned}$$

By $\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}} = \emptyset$, the following holds:

$$\forall i \in \mathcal{J}^{+}_{\mathbf{w}_d}.\,[x_i = -1 \wedge \mathrm{a}_i \notin \mathcal{P}_{\mathbf{x},\text{facts}}]$$

So all rules of the form $\mathrm{d} \leftarrow \mathrm{a}_i$ for $i \in \mathcal{J}^{+}_{\mathbf{w}_d}$ have false bodies.

And by $\mathcal{J}^{-,+}_{\mathbf{w}_d,\mathbf{x}} = \mathcal{J}^{-}_{\mathbf{w}_d}$, the following holds:

$$\forall i \in \mathcal{J}^{-}_{\mathbf{w}_d}.\,[x_i = 1 \wedge \mathrm{a}_i \in \mathcal{P}_{\mathbf{x},\text{facts}}]$$

So for all $i \in \mathcal{J}^{-}_{\mathbf{w}_d}$, `not` a_i is false and thus all rules of the form $\mathrm{d} \leftarrow$ `not` a_i have false bodies. Therefore, the body of every rule in $\mathcal{P}_{\mathbf{w}_d,\text{rules}}$ is false, and the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and $\mathcal{P}_{\mathbf{w}_d,\text{rules}}$ does not have d in its answer set (i.e. d is false).

Together, these two cases establish that the bivalent logical interpretation b_d of the disjunctive semi-symbolic node has the same truth value as the rule head d in the combined program of $\mathcal{P}_{\mathbf{x},\text{facts}}$ and the rules $\mathcal{P}_{\mathbf{w}_d,\text{rules}}$ from Equation (4.24), whether $f_{\mathbf{w}_d}(\mathbf{x}) \geq 6$ or $f_{\mathbf{w}_d}(\mathbf{x}) = -6$. ■

Proof of Theorem 4.2.3. Following the same derivation as in the proof of Theorem 4.2.1, the raw output of the translated conjunctive node can be expressed in terms of the four mutually exclusive

subsets as shown in Equation (A.1):

$$f_{\mathbf{w}_c}(\mathbf{x}) = 6\left(1 - 2|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - 2|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}|\right) \tag{A.5}$$

Two cases are considered.

Case 1: h_c is in the answer set (i.e. h_c is true). It must be proven that $f_{\mathbf{w}_c}(\mathbf{x}) = 6 > 0$ such that the conjunctive node's bivalent logical interpretation $b_c = \top$.

For h_c to be in the answer set, the body of the rule must be satisfied. That is:

$$\left(\forall i \in \mathcal{J}^{+}_{\mathbf{w}_c}.\mathrm{a}_i \text{ is true}\right) \wedge \left(\forall i \in \mathcal{J}^{-}_{\mathbf{w}_c}.\texttt{not } \mathrm{a}_i \text{ is true}\right)$$

Combined with the translation defined in Equation (4.36), this gives:

$$\begin{aligned}&\left(\forall i \in \mathcal{J}^{+}_{\mathbf{w}_c}.x_i = 1\right) \wedge \left(\forall i \in \mathcal{J}^{-}_{\mathbf{w}_c}.x_i = -1\right)\\ &\Rightarrow \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}} = \emptyset \wedge \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}} = \emptyset\end{aligned}$$

Substituting the values into Equation (A.5) gives:

$$\begin{aligned}&f_{\mathbf{w}_c}(\mathbf{x}) = 6\left(1 - 2\cdot 0 - 2\cdot 0\right) = 6 > 0\\ &\Rightarrow b_c = \top\end{aligned}$$

Case 2: h_c is not in the answer set (i.e. h_c is false). It must be proven that $f_{\mathbf{w}_c}(\mathbf{x}) \leq -6 < 0$ such that the conjunctive node's bivalent logical interpretation $b_c = \bot$.

For h_c to be not in the answer set, the body of the rule must not be satisfied. That is, at least one of the disjuncts in the following disjunction must hold:

$$\left(\exists i \in \mathcal{J}^{+}_{\mathbf{w}_c}.\mathrm{a}_i \text{ is false}\right) \vee \left(\exists i \in \mathcal{J}^{-}_{\mathbf{w}_c}.\texttt{not } \mathrm{a}_i \text{ is false}\right)$$

In terms of the four mutually exclusive subsets:

- If a_i is false for some $i \in \mathcal{J}^{+}_{\mathbf{w}_c}$, then $x_i = -1$, meaning $i \in \mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}$.
- If $\texttt{not } \mathrm{a}_i$ is false for some $i \in \mathcal{J}^{-}_{\mathbf{w}_c}$, then a_i is true and $x_i = 1$, meaning $i \in \mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}$.

Therefore, the following inequality holds:

$$|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| + |\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| \geq 1$$

Substituting into Equation (A.5):

$$\begin{aligned}1 - 2|\mathcal{J}^{+,-}_{\mathbf{w}_c,\mathbf{x}}| - 2|\mathcal{J}^{-,+}_{\mathbf{w}_c,\mathbf{x}}| &\leq 1 - 2 = -1\\ \Rightarrow f_{\mathbf{w}_c}(\mathbf{x}) &\leq -6 < 0\\ \Rightarrow b_c &= \bot\end{aligned}$$

Together, these two cases establish that the rule head h_c's truth value equals the bivalent logical interpretation b_c of the translated conjunctive semi-symbolic node, whether h_c is true or false under some given facts. ■

Proof of Theorem 4.2.4. Following the same derivation as in the proof of Theorem 4.2.2, the raw output of the translated disjunctive node can be expressed in terms of the four mutually exclusive subsets as shown in Equation (A.3):

$$f_{\mathbf{w}_d}(\mathbf{x}) = 6\left(2|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| - 1\right) \tag{A.6}$$

Two cases are considered.

Case 1: h_d is in the answer set (i.e. h_d is true). It must be proven that $f_{\mathbf{w}_d}(\mathbf{x}) \geq 6 > 0$ such that the disjunctive node's bivalent logical interpretation $b_d = \top$.

For h_d to be in the answer set, at least one rule body must be satisfied. That is:

$$\left(\exists i \in \mathcal{J}^{+}_{\mathbf{w}_d}.\mathrm{a}_i \text{ is true}\right) \vee \left(\exists i \in \mathcal{J}^{-}_{\mathbf{w}_d}.\texttt{not } \mathrm{a}_i \text{ is true}\right)$$

In terms of the four mutually exclusive subsets:

- If a_i is true for some $i \in \mathcal{J}^{+}_{\mathbf{w}_d}$, then $x_i = 1$, meaning $i \in \mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}$.
- If $\texttt{not } \mathrm{a}_i$ is true for some $i \in \mathcal{J}^{-}_{\mathbf{w}_d}$, then a_i is false and $x_i = -1$, meaning $i \in \mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}$.

Since at least one of the conditions above must hold, the following inequality holds:

$$|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + |\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| \geq 1$$

Substituting into Equation (A.6):

$$\begin{aligned} 2|\mathcal{J}^{+,+}_{\mathbf{w}_d,\mathbf{x}}| + 2|\mathcal{J}^{-,-}_{\mathbf{w}_d,\mathbf{x}}| - 1 &\geq 2 - 1 = 1 \\ \Rightarrow f_{\mathbf{w}_d}(\mathbf{x}) &\geq 6 > 0 \\ \Rightarrow b_d &= \top \end{aligned}$$

Case 2: h_d is not in the answer set (i.e. h_d is false). It must be proven that $f_{\mathbf{w}_d}(\mathbf{x}) = -6 < 0$ such that the disjunctive node's bivalent logical interpretation $b_d = \bot$.

For h_d to be not in the answer set, all rule bodies must not be satisfied. That is:

$$\left(\forall i \in \mathcal{J}^{+}_{\mathbf{w}_d}.\mathrm{a}_i \text{ is false}\right) \wedge \left(\forall i \in \mathcal{J}^{-}_{\mathbf{w}_d}.\texttt{not } \mathrm{a}_i \text{ is false}\right)$$

Combined with the translation defined in Equation (4.41), this gives:

$$\left(\forall i \in \mathcal{J}^{+}_{\mathbf{w}_d}.x_i = -1\right) \wedge \left(\forall i \in \mathcal{J}^{-}_{\mathbf{w}_d}.x_i = 1\right)$$

Using the four mutually exclusive subsets, the following equalities hold:

$$\begin{aligned}&\mathcal{J}^{+,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} = \mathcal{J}^{+}_{\mathbf{w}_\mathrm{d}} \wedge \mathcal{J}^{-,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} = \mathcal{J}^{-}_{\mathbf{w}_\mathrm{d}}\\ &\Rightarrow \mathcal{J}^{+,+}_{\mathbf{w}_\mathrm{d},\mathbf{x}} = \emptyset \wedge \mathcal{J}^{-,-}_{\mathbf{w}_\mathrm{d},\mathbf{x}} = \emptyset\end{aligned}$$

Substituting the values into Equation (A.6) gives:

$$\begin{aligned}&f_{\mathbf{w}_\mathrm{d}}(\mathbf{x}) = 6\,(2 \cdot 0 + 2 \cdot 0 - 1) = -6 < 0\\ &\Rightarrow b_\mathrm{d} = \bot\end{aligned}$$

Together, these two cases establish that the rule head h_d's truth value equals the bivalent logical interpretation b_d of the translated disjunctive semi-symbolic node, whether h_d is true or false under some given facts. ■

Appendix B

Classification Experiments

B.1 CUB Dataset/subsets

B.1.1 Data Pre-processing

Algorithm 8 shows the pseudo-code of the data pre-processing procedure used in Chapter 5 and also in Concept Bottleneck Models [88]. After such pre-processing, all samples of the same class would have the exact same attribute encoding. In the Concept Bottleneck Models paper, the encoding for each class is computed from the training set and also used as the encoding for the validation set and test set as well. Although it is an odd decision to change the validation set and test set using results from the training set, the data pre-processing does ensure attribute label consistency both within each class and across samples and in all 3 sets. This decision is followed in the pre-processing for a fair comparison. The N values chosen for CUB-3, -10, -15, -20, -25, -50, -100, and -200 are 1, 2, 3, 3, 3, 5, 7, and 10 respectively.

B.2 TMC Dataset/subsets

B.2.1 Data Pre-processing

The mutual information (MI) of each attribute with respect to the output label combinations is calculated, and attributes with MI value less than a threshold t are filtered out. The t values chosen for TMC-3, -5, -10, -15, and -22 are 0.01, 0.02, 0.04, 0.04 and 0.05 respectively. The following provides the mathematical formulae to calculate the mutual information.

Mutual information of two discrete random variables X and Y can be defined as:

$$\mathrm{I}(X;Y) \equiv \mathrm{H}(X) - \mathrm{H}(X|Y) \tag{B.1}$$

where $\mathrm{H}(X)$ and $\mathrm{H}(Y)$ are entropies, and $\mathrm{H}(X|Y)$ is conditional entropy. $\mathrm{H}(X)$ is calculated as:

$$\mathrm{H}(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{B.2}$$

And the conditional entropy $\mathrm{H}(X|Y)$ is calculated as:

Algorithm 8 Data pre-processing for CUB dataset.

Input: Dataset D, Total number of classes n_c, Total number of attributes in CUB n_a
Output: N – number of classes an attribute should be present consistently

```
/* Count for each attribute in each class how many times it is present/not present */
class_attribute_count ← n_c × n_a × 2 array of all zeros
for d ∈ D do
    for Attribute a ∈ d.attributes do
        if a.is_present == 0 ∧ a.certainty == 1 then
            /* Ignore attribute that is not present because not visible */
            continue
        class_attribute_count[d.class][a.attribute_id][a.is_present] += 1
/* For each class, for each attribute, which presence label (present or not present) is less frequent */
class_attribute_min_presence ← arg min(class_attribute_count, axis = 2)
/* For each class, for each attribute, which presence label (present or not present) is more frequent */
class_attribute_max_presence ← arg max(class_attribute_count, axis = 2)
/* Find where most and least frequent, set most frequent to 1 */
equal_count_index ← indices where class_attribute_min_presence == class_attribute_max_presence
class_attribute_max_presence[equal_count_index] ← 1
/* Count number of times each attribute is mostly present for a class */
attribute_class_count ← sum(class_attribute_max_presence, axis = 0)
/* Select the attributes that are present most of the time on a class level, in at least N classes */
mask ← indices where attribute_class_count ≥ N
/* Get the attribute median */
class_attribute_median ← class_attribute_max_presence[:, mask]
```

$$\mathrm{H}(X|Y) = -\sum_{y\in\mathcal{Y}} p(y) \sum_{x\in\mathcal{X}} p(x|y)\log p(x|y) \tag{B.3}$$

For each attribute X_i, the mutual information of it with respect to the output label combination Y is calculated as the following, using Equation B.1, B.2 and B.3:

$$\mathrm{I}(X_i;Y) = -\sum_{x_i\in\mathcal{X}_i} p(x_i)\log p(x_i) - \left(-\sum_{y\in\mathcal{Y}} p(y) \sum_{x_i\in\mathcal{X}_i} p(x_i|y)\log p(x_i|y)\right) \tag{B.4}$$

Since each attribute only has two outcomes: present (1) or not present (0), Equation (B.4) is equivalent to:

$$\begin{aligned}\mathrm{I}(X_i;Y) = &-\Big(p(X_i=1)\log p(X_i=1) + p(X_i=0)\log p(X_i=0)\Big) \\ &-\left(-\sum_{y\in\mathcal{Y}} p(y)\Big(p(X_i=1|y)\log p(X_i=1|y) + p(X_i=0|y)\log p(X_i=0|y)\Big)\right)\end{aligned} \tag{B.5}$$

B.3 Logical Interpretation Examples

Binary Classification

Below is an example of the logical interpretation of a neural DNF model in the the dataset **Monk** [162].

```
% t is the target class predicate.
% a_i is the input feature.
t :- a_1, a_4.   t :- a_2, a_5.
t :- a_0, a_3.
t :- not a_11, not a_12, not a_13.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **BCC** [122], where threshold-learning predicate invention method is used.

```
% Invented predicates defined by the threshold-learning
% predicate invention method.
a_2 = feature_0 > -269.8374328613281
a_3 = feature_0 > 186.82131958007812
a_6 = feature_1 > 58.69133377075195
a_6 = feature_1 > 58.69133377075195
a_8 = feature_2 > 197.28598022460938
a_14 = feature_3 > -13.131363868713379
a_28 = feature_7 > 4.719961166381836

% Rules
% t is the target class predicate.
% a_i are invented predicates shown above.
t :- a_2, not a_3, not a_6.
t :- a_8, a_14.
t :- a_2, not a_6, a_28.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **Mushroom** [67].

```
% t is the target class predicate.
% a_i are input features.
t :- a_50, a_97, a_103.
t :- a_46, a_89, not a_105, not a_109.
t :- not a_21, not a_24, not a_26.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **CDC** diabetes [152], where threshold-learning predicate invention method is used.

```
% Invented predicates defined by the threshold-learning
% predicate invention method.
a_2 = feature_0 > 74.53063201904297
```

```
a_5 = feature_0 > 58.61220932006836
a_6 = feature_0 > -33.472694396972656
a_7 = feature_0 > 102.41500854492188
a_8 = feature_1 > 22.021968841552734
a_12 = feature_1 > 16.420833587646484
a_15 = feature_1 > 19.272249221801758
a_18 = feature_2 > 45.10814666748047
a_25 = feature_3 > 51.938621520996094

% Rules
% t is the target class predicate.
% a_i with i in [0, 63] are invented predicates shown above.
% a_i with i in [63, 91] are binary input features.
t :- a_35.
t :- a_38.
t :- a_52.
t :- a_47.
t :- a_43.
t :- not a_37.
t :- a_39.
t :- a_45.
t :- a_7.
t :- a_25.
t :- a_18.
t :- a_2, a_12, not a_32, not a_41.
t :- a_2, a_15, not a_32.
t :- not a_34, a_36, not a_42, not a_59.
t :- a_8, a_36, not a_42, not a_59.
t :- a_36, not a_42, a_55.
t :- a_48.
t :- not a_41, not a_42, not a_59.
t :- not a_34, a_36, not a_51.
t :- not a_49, a_53.
t :- a_5, not a_42, not a_50, not a_51, not a_57.
t :- not a_6, not a_59.
t :- a_40, not a_51.
t :- a_36, a_53.
```

Multi-class Classification

Below is an example of the logical interpretation of a neural DNF-MT model in the dataset **Car** [22].

```
% conj_i are the output predicates of the conjunctive layer.
```

```
% a_i are the input features.
conj_0 :- not a_13, not a_14.
conj_1 :- a_2, not a_5, not a_12, not a_17, a_20.
conj_1 :- a_2, not a_12, a_18.
conj_1 :- not a_7, not a_12, a_18.
conj_1 :- a_1, not a_12, a_18.
conj_1 :- a_1, not a_5, not a_16, a_20.
conj_3 :- not a_0, not a_3, a_5, not a_8, a_14, not a_17,
    a_20.
conj_3 :- a_1, a_5, not a_9, not a_12, not a_17, a_20.
conj_6 :- not a_9, not a_10, not a_13, a_17.
conj_7 :- not a_2, not a_5, a_9, not a_14, a_16, not a_18.
conj_7 :- not a_1, not a_6, not a_11, not a_14, a_16,
    not a_18.
conj_8 :- a_2, a_5, not a_12, not a_19, not a_20.
conj_8 :- a_2, a_5, a_13, not a_19.
conj_9 :- not a_7, not a_17, not a_19.
conj_9 :- not a_3, not a_7, not a_19.
conj_9 :- not a_0, not a_3, not a_19, not a_20.
conj_12 :- not a_20.
conj_13 :- a_3, not a_5, not a_6.
conj_14 :- a_1, a_5, not a_12, not a_19, not a_20.
conj_15 :- a_1, not a_7, not a_12, not a_17, a_18.
conj_16 :- not a_0.
conj_17 :- a_0, not a_7, not a_12, not a_17, not a_19.
conj_21 :- a_1, a_6, not a_12, not a_19.
conj_22 :- not a_0, not a_3, not a_4, not a_7, not a_8,
    not a_12, not a_17, not a_19, not a_20.
conj_24 :- not a_5, not a_6, a_17, a_20.
% During inference, given an input, the conjunctions that
% are activated are first computed.
% Then the ProbLog rule is generated.
% For example:
% Input (tensor):
[-1., -1.,  1., -1., -1., -1., -1.,  1., -1.,  1., -1.,
-1., -1., -1., 1., -1.,  1., -1., -1., -1.,  1.]
% Input (translated to predicates):
[a_2, a_7, a_9, a_14, a_16, a_20]
% Conjunctions that are activated:
[conj_1, conj_16]
% ProbLog rule:
0.806::class_0 ; 0.020::class_1 ; 0.171::class_2 ;
0.003::class_3 :- conj_1, conj_16.
```

```
% Final prediction: class_0, ground truth: class_0
```

Below is an example of the logical interpretation of a neural DNF-MT model in the dataset **Covertype** [21], where an MLP is used as a predicate inventor.

```
% conj_i are the output predicates of the conjunctive layer.
% a_i are the invented predicates from the MLP predicate
% inventor.
conj_0 :- not a_14.
conj_1 :- not a_6, a_27, not a_57.
conj_2 :- a_1.
conj_3 :- not a_46, a_52.
conj_6 :- not a_26.
conj_7 :- a_42.
conj_10 :- a_24.
conj_13 :- a_0.
conj_16 :- a_12, a_36, not a_51.
conj_17 :- not a_3.
conj_18 :- not a_50, a_52.
conj_19 :- not a_18, a_62.
conj_19 :- a_34.
conj_20 :- not a_57.
conj_21 :- a_12, a_17.
conj_22 :- a_0, not a_41.
conj_23 :- not a_49, not a_61.
conj_24 :- a_20.
conj_25 :- not a_21, not a_26.
conj_26 :- a_63.
conj_27 :- not a_51, a_56.
conj_28 :- not a_26, a_39.
conj_29 :- a_39.
conj_30 :- not a_26, a_48.
conj_32 :- not a_40.
conj_33 :- a_44.
conj_34 :- a_10.
conj_35 :- not a_44.
conj_36 :- a_33.
conj_37 :- not a_42.
conj_38 :- not a_44.
conj_40 :- a_29.
conj_41 :- not a_55.
conj_42 :- not a_10, not a_58.
conj_43 :- not a_61, a_62.
conj_44 :- not a_48.
```

```
conj_45 :- a_41, not a_50.
conj_46 :- a_31.
conj_47 :- a_43.
conj_48 :- a_12.
conj_50 :- not a_54.
conj_51 :- a_19, not a_59.
conj_52 :- not a_1, not a_2, a_9, not a_51.
conj_53 :- not a_15, not a_22, not a_29, not a_38, not a_43.
conj_54 :- a_9, not a_15, a_37.
conj_55 :- a_49.
conj_56 :- not a_49, not a_53.
conj_57 :- a_36, not a_37.

% During inference, given an input, the invented predicates
% from the MLP predicate inventor are first computed.
% Then the activated conj_i are identified.
% Finally, the ProbLog rule is generated.
% For example:
% Input:
[ 1.3166, -0.5331, -0.2809, -0.1949, -0.4017,  0.2064,  1.0031,
-0.0667, -0.2774,  1.0000, -1.0000,  1.0000,  1.0000,  1.0000,
-1.0000, -1.0000]
% Invented predicates that are true:
[a_0, a_1, a_2, a_3, a_7, a_8, a_9, a_10, a_11, a_14, a_15, a_17,
a_22, a_26, a_27, a_28, a_29, a_31, a_32, a_33, a_34, a_38, a_39,
a_41, a_43, a_44, a_47, a_48, a_49, a_51, a_53, a_54, a_58, a_59,
a_60, a_61, a_63]
% Conjunctions that are activated:
[conj_1, conj_2, conj_13, conj_19, conj_20, conj_26, conj_29,
conj_32, conj_33, conj_34, conj_36, conj_37, conj_40, conj_41,
conj_45, conj_46, conj_47, conj_55]
% ProbLog rule:
0.005::class_0 ; 0.047::class_1 ; 0.000::class_2 ;
0.000::class_3 ; 0.128::class_4 ; 0.001::class_5 ;
0.820::class_6 :- conj_1, conj_2, conj_13, conj_19, conj_20,
conj_26, conj_29, conj_32, conj_33, conj_34, conj_36, conj_37,
conj_40, conj_41, conj_45, conj_46, conj_47, conj_55.
% Final prediction: class_6, ground truth: class_6
```

Multi-label Classification

The datasets in this section are Boolean Network [82] datasets, with ground-truth logic programs. Boolean Network models the state transitions of genes from time t to $t+1$, and the learning of such

a dynamic system is treated as a multi-label classification problem: the inputs are gene states at time t and the labels are the gene states at time $t+1$.

Below is an example of the logical interpretation of a neural DNF model in the dataset **ARA** [25].

```
% l_i are the target labels.
% a_i are inputs.
l_0 :- a_1, a_6.
l_0 :- a_0, a_4, a_13, a_14.
l_0 :- a_0, a_9, a_13, a_14.
l_1 :- a_1.
l_2 :- not a_4, not a_12.
l_3 :- not a_5.
l_4 :- a_6, not a_9.
l_4 :- a_3, not a_9.
l_4 :- not a_9, not a_12.
l_5 :- not a_6.
l_6 :- not a_12.
l_6 :- not a_5.
l_7 :- not a_12.
l_8 :- a_8, not a_9.
l_8 :- a_8, not a_14.
l_9 :- not a_7, not a_12.
l_9 :- a_6, a_8.
l_9 :- a_6, not a_11.
l_9 :- not a_4, a_6.
l_9 :- a_6, not a_10.
l_12 :- not a_4, a_5, not a_6.
l_13 :- a_6, a_9.
l_13 :- a_0, a_6.
l_14 :- a_6.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **Budding** [94].

```
% l_i are the target labels.
% a_i are inputs.
l_1 :- a_0.
l_2 :- a_2, not a_8.
l_2 :- a_1, not a_8.
l_2 :- a_1, a_2.
l_3 :- a_3, not a_8.
l_3 :- a_1, a_3.
l_3 :- a_1, not a_8.
l_4 :- a_2.
```

```
l_5 :- not a_4, not a_8, not a_10, a_11.
l_5 :- not a_4, not a_6, not a_8, a_10.
l_5 :- not a_4, not a_6, a_10, a_11.
l_5 :- not a_6, not a_8, a_10, a_11.
l_5 :- a_5, not a_6, not a_8, a_11.
l_5 :- not a_4, a_5, not a_6, not a_8.
l_5 :- not a_4, a_5, not a_8, a_10.
l_5 :- not a_4, a_5, a_10, a_11.
l_5 :- a_5, not a_8, a_10, a_11.
l_5 :- not a_4, a_5, not a_6, a_11.
l_5 :- a_5, not a_6, a_10, a_11.
l_5 :- a_5, not a_6, not a_8, a_10.
l_5 :- not a_4, a_5, not a_6, a_10.
l_6 :- a_3, not a_5, not a_10.
l_6 :- not a_5, a_6, not a_10.
l_6 :- a_3, a_6, not a_10.
l_6 :- a_3, not a_5, a_6.
l_7 :- not a_4, a_7, not a_8, a_10.
l_7 :- not a_4, not a_6, not a_8, a_10.
l_7 :- not a_4, not a_6, a_7, a_10.
l_7 :- not a_4, not a_6, a_7, not a_8.
l_7 :- not a_6, a_7, not a_8, a_10.
l_8 :- not a_5, a_6, not a_7, a_8.
l_8 :- a_6, not a_7, a_9, not a_10.
l_8 :- not a_5, not a_7, a_9, not a_10.
l_8 :- not a_5, a_6, not a_7, a_9.
l_8 :- a_6, not a_7, a_8, a_9.
l_8 :- not a_5, a_6, not a_7, not a_10.
l_8 :- not a_7, a_8, a_9, not a_10.
l_8 :- not a_5, not a_7, a_8, a_9.
l_8 :- a_6, not a_7, a_8, not a_10.
l_8 :- not a_5, not a_7, a_8, not a_10.
l_8 :- not a_5, a_8, a_9, not a_10.
l_8 :- not a_5, a_6, a_9, not a_10.
l_8 :- not a_5, a_6, a_8, not a_10.
l_8 :- not a_5, a_6, a_8, a_9.
l_8 :- a_6, a_8, a_9, not a_10.
l_9 :- a_6.
l_9 :- a_8.
l_10 :- a_8.
l_10 :- a_9.
l_11 :- a_9, a_10.
l_11 :- not a_8, a_9.
```

```
l_11 :- not a_8, a_10.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **Fission** [35].

```
% l_i are the target labels.
% a_i are inputs.
l_1 :- a_0.
l_2 :- a_2, not a_3, a_5, not a_9.
l_2 :- not a_1, not a_3, a_5, not a_9.
l_2 :- not a_1, a_2, not a_3, a_5.
l_2 :- not a_1, a_2, not a_3, not a_9.
l_2 :- not a_1, a_2, a_5, not a_9.
l_3 :- not a_2, a_3, not a_4.
l_3 :- a_3, not a_4, not a_7.
l_3 :- not a_2, a_3, not a_7.
l_3 :- not a_2, not a_4, not a_7.
l_4 :- not a_1, a_4, a_5, not a_9.
l_4 :- not a_1, not a_3, not a_9.
l_4 :- not a_1, not a_3, not a_4, not a_5.
l_4 :- not a_3, a_4, a_5, a_9.
l_5 :- a_7.
l_6 :- a_3, not a_5.
l_6 :- not a_5, a_6.
l_6 :- a_3, a_6.
l_7 :- a_9.
l_8 :- not a_3, a_8.
l_8 :- not a_3, a_5.
l_8 :- a_5, a_8.
l_9 :- not a_2, not a_4, a_6, not a_7, not a_8, a_9.
```

Below is an example of the logical interpretation of a neural DNF model in the dataset **MAM** [52].

```
% l_i are the target labels.
% a_i are inputs.
l_0 :- a_0.
l_1 :- not a_2, a_3.
l_2 :- not a_0, a_5, not a_9.
l_2 :- not a_0, not a_1, not a_4, not a_9.
l_3 :- not a_2, a_5, not a_9.
l_3 :- not a_2, not a_4, not a_9.
l_4 :- not a_2, a_3, not a_6, not a_8.
l_4 :- not a_2, a_3, not a_6, not a_7.
l_4 :- not a_2, a_4, not a_6, not a_8.
```

```
12 l_4 :- not a_2, a_4, not a_6, not a_7.
13 l_5 :- not a_0, not a_4, a_5, not a_9.
14 l_5 :- not a_0, not a_1, a_5, not a_9.
15 l_5 :- not a_0, not a_1, not a_4, not a_9.
16 l_6 :- a_9.
17 l_7 :- not a_8.
18 l_7 :- a_7, a_9.
19 l_7 :- a_6, a_7.
20 l_7 :- a_4, a_7.
21 l_8 :- a_6.
22 l_8 :- a_5, a_9.
23 l_8 :- not a_4, not a_9.
24 l_9 :- not a_6, not a_8
```

Appendix C

Neural DNF-MT Model for Reinforcement Learning

This appendix provides more details on the methodology and experiments in Chapter 6, including post-training algorithms, the model architectures, training hyperparameters, and more examples of logical interpretation of neural DNF-MT actors.

C.1 Post-training Algorithms

The algorithms for each stage in the post-training process in Section 6.2.3 are listed below:

Algorithm 9 Pruning of a Trained Neural DNF-MT Actor

Input: Trained neural DNF-MT actor with weights $\mathbf{w}$, set of representative states $\mathcal{S}$ (can be all possible states if enumerable, or the states observed during training if not), threshold τ_{prune}, policy type $\in \{\text{deterministic}, \text{stochastic}\}$
Output: Pruned weights $\mathbf{w}$

procedure IsRemovalAcceptable($\mathbf{w}, \mathbf{w}', \mathcal{S}$)
  if policy type is deterministic **then**
    return $\forall s \in \mathcal{S}.\ \arg\max \pi_{\mathbf{w}}(s) = \arg\max \pi_{\mathbf{w}'}(s)$ ▷ Trajectory preserved
  else
    return $\forall s \in \mathcal{S}, \forall j.\ |\pi_{\mathbf{w}}(s)_j - \pi_{\mathbf{w}'}(s)_j| \leq \tau_{\text{prune}}$ ▷ Action probabilities preserved
procedure PruneEdges($\mathbf{w}_{\text{layer}}, \mathcal{S}$)
  removed $\leftarrow$ **false**
  for each edge weight w_i in $\mathbf{w}_{\text{layer}}$ **do**
    $\mathbf{w}' \leftarrow \mathbf{w}$ with w_i set to 0
    if IsRemovalAcceptable($\mathbf{w}, \mathbf{w}', \mathcal{S}$) **then**
      $\mathbf{w} \leftarrow \mathbf{w}'$ ▷ Accumulate the removal
      removed $\leftarrow$ **true**
  return *removed*
repeat
  $r_{\text{d}} \leftarrow$ PruneEdges($\mathbf{w}_{\text{d}}, \mathcal{S}$) ▷ Conjunction-to-disjunction edges first
  $r_{\text{c}} \leftarrow$ PruneEdges($\mathbf{w}_{\text{c}}, \mathcal{S}$) ▷ Then input-to-conjunction edges
  Remove any unconnected conjunctive or disjunctive nodes from $\mathbf{w}$
until $\neg r_{\text{d}} \wedge \neg r_{\text{c}}$ ▷ Stop when a full pass yields no removals
return $\mathbf{w}$

Algorithm 10 Thresholding of a Trained Neural DNF-MT Actor

Input: Pruned neural DNF-MT actor with weights $\mathbf{w}$, set of representative states $\mathcal{S}$ (can be all possible states if enumerable, or the states observed during training if not), threshold τ_{prune}, step size s, policy type $\in \{\text{deterministic}, \text{stochastic}\}$

Output: Thresholded weights $\hat{\mathbf{w}} \in \{-6, 0, 6\}$, with tanh activations replaced by the step function $h(x) = 2 \cdot \mathbb{1}(x > 0) - 1$ on thresholded nodes that have at least one non-zero weight

procedure THRESHOLD($\mathbf{w}_{\text{layer}}, \tau$)
  $\hat{w}_{kij} \leftarrow 6 \cdot \text{sign}(w_{kij}) \cdot \mathbb{1}(|w_{kij}| \geq \tau)$ for each weight in $\mathbf{w}_{\text{layer}}$
  return $\hat{\mathbf{w}}_{\text{layer}}$
procedure ISTHRESHOLDACCEPTABLE($\mathbf{w}, \hat{\mathbf{w}}, \mathcal{S}$) ▷ Same checks as in Algorithm 9
  if policy type is deterministic **then**
    return $\forall s \in \mathcal{S}.\ \arg\max \pi_{\mathbf{w}}(s) = \arg\max \pi_{\hat{\mathbf{w}}}(s)$
  else
    return $\forall s \in \mathcal{S}, \forall j.\ |\pi_{\mathbf{w}}(s)_j - \pi_{\hat{\mathbf{w}}}(s)_j| \leq \tau_{\text{prune}}$
procedure FINDBESTTHRESHOLD($\mathbf{w}_{\text{target}}, \mathbf{w}, \mathcal{S}$)
  $\tau^* \leftarrow 0$
  for $\tau \in [0, \max|\mathbf{w}_{\text{target}}|, s]$ **do**
    $\hat{\mathbf{w}}_{\text{target}} \leftarrow$ THRESHOLD($\mathbf{w}_{\text{target}}, \tau$)
    $\hat{\mathbf{w}} \leftarrow \mathbf{w}$ with $\mathbf{w}_{\text{target}}$ replaced by $\hat{\mathbf{w}}_{\text{target}}$
    if ISTHRESHOLDACCEPTABLE($\mathbf{w}, \hat{\mathbf{w}}, \mathcal{S}$) **and** $\tau > \tau^*$ **then**
      $\tau^* \leftarrow \tau$ ▷ Prefer the largest acceptable threshold
  return τ^*
if policy type is stochastic **then** ▷ Threshold the conjunctive layer only (2.a)
  $\tau_c^* \leftarrow$ FINDBESTTHRESHOLD($\mathbf{w}_c, \mathbf{w}, \mathcal{S}$)
  $\mathbf{w}_c \leftarrow$ THRESHOLD($\mathbf{w}_c, \tau_c^*$)
else ▷ Threshold both layers with a single τ (2.b)
  $\tau^* \leftarrow 0$
  for $\tau \in [0, \max(\max|\mathbf{w}_c|, \max|\mathbf{w}_d|), s]$ **do**
    $\hat{\mathbf{w}}_c \leftarrow$ THRESHOLD($\mathbf{w}_c, \tau$), $\hat{\mathbf{w}}_d \leftarrow$ THRESHOLD($\mathbf{w}_d, \tau$)
    $\hat{\mathbf{w}} \leftarrow \mathbf{w}$ with both layers replaced
    if ISTHRESHOLDACCEPTABLE($\mathbf{w}, \hat{\mathbf{w}}, \mathcal{S}$) **and** $\tau > \tau^*$ **then**
      $\tau^* \leftarrow \tau$
  $\mathbf{w}_c \leftarrow$ THRESHOLD($\mathbf{w}_c, \tau^*$), $\mathbf{w}_d \leftarrow$ THRESHOLD($\mathbf{w}_d, \tau^*$)
Replace tanh with the step function h on every thresholded node having at least one non-zero weight
return $\mathbf{w}$

Algorithm 11 Post-training Process for Stochastic Policy Translation

Input: Trained neural DNF-MT model with conjunctive nodes $\{c_1, \ldots, c_{C'}\}$
Output: ProbLog annotated disjunctions

procedure CONDENSELOGICALLYEQUIVALENTNODES
  for each pair of conjunctive nodes c_i, c_j **do**
    if c_i and c_j have identical weight patterns **then**
      Mark c_i and c_j as logically equivalent
procedure SIMPLIFYRULESBASEDONOBSERVATIONS
  Note: This procedure is optional if input enumeration is infeasible
  Compute all possible inputs to conjunctive layer
  $\mathcal{A} \leftarrow$ all unique activations of conjunctive layer
  for each conjunctive node c_i **do**
    if c_i is always true or always false across all activations in $\mathcal{A}$ **then**
      Mark c_i as redundant and not needed in any rule's body in the ProbLog program
procedure GENERATECONJUNCTIVERULES
  for each conjunctive node c_i after condensation and simplification **do**
    $\mathcal{I}_i \leftarrow \{k \mid w'_{cki} \neq 0\}$
    $\mathcal{I}_i^+ \leftarrow \{k \in \mathcal{I}_i \mid w'_{cki} = 6\}$
    $\mathcal{I}_i^- \leftarrow \{k \in \mathcal{I}_i \mid w'_{cki} = -6\}$
    Generate rule: $conj_i \leftarrow \bigwedge_{k \in \mathcal{I}_i^+} input_k, \bigwedge_{k \in \mathcal{I}_i^-} (\backslash + input_k)$
procedure GENERATEPROBLOGANNOTATEDDISJUNCTIONS
  for each unique conjunctive activation $\mathbf{c}^{(m)} \in \{-1, 1\}^{C'}$ **do**
    Compute mutex-tanh output $\tilde{\mathbf{y}}^{(m)}$
    $p_j \leftarrow (\tilde{y}_j^{(m)} + 1)/2$ for each action j
    $\mathcal{C}^{(m)+} \leftarrow \{i | c_i^{(m)} = 1\}$
    $\mathcal{C}^{(m)-} \leftarrow \{i | c_i^{(m)} = -1\}$
    Generate rule: $p_1 :: action_1; \ldots; p_N :: action_N \leftarrow \bigwedge_{i \in \mathcal{C}^{(m)+}} conj_i, \bigwedge_{i \in \mathcal{C}^{(m)-}} (\backslash + conj_i)$
CONDENSELOGICALLYEQUIVALENTNODES
SIMPLIFYRULESBASEDONOBSERVATIONS
GENERATECONJUNCTIVERULES
GENERATEPROBLOGANNOTATEDDISJUNCTIONS

Algorithm 12 Post-training Process for Deterministic Policy Translation

Input: Trained neural DNF-MT model with pruned and thresholded weights, with conjunctive nodes $\{c_1, \ldots, c_{C'}\}$ and disjunctive nodes $\{d_1, \ldots, d_N\}$

Output: ASP program

procedure GENERATECONJUNCTIVERULES
  for each conjunctive node c_i **do**
    $\mathcal{I}_i \leftarrow \{k \mid w'_{cki} \neq 0\}$
    $\mathcal{I}_i^+ \leftarrow \{k \in \mathcal{I}_i \mid w'_{cki} = 6\}$
    $\mathcal{I}_i^- \leftarrow \{k \in \mathcal{I}_i \mid w'_{cki} = -6\}$
    Generate rule: $conj_i \leftarrow \bigwedge_{k \in \mathcal{I}_i^+} input_k, \bigwedge_{k \in \mathcal{I}_i^-} (\text{not } input_k)$
procedure GENERATEDISJUNCTIVERULES
  for each disjunctive node d_j **do**
    $\mathcal{C}_j \leftarrow \{i \in \{1..C'\} \mid w'_{dij} \neq 0\}$
    $\mathcal{C}_j^+ \leftarrow \{i \in \mathcal{C}_j \mid w'_{dij} = 6\}$
    $\mathcal{C}_j^- \leftarrow \{i \in \mathcal{C}_j \mid w'_{dij} = -6\}$
    for each $i \in \mathcal{C}_j^+$ **do**
      Generate rule: $disj_j \leftarrow conj_i$
    for each $i \in \mathcal{C}_j^-$ **do**
      Generate rule: $disj_j \leftarrow \text{not } conj_i$
GENERATECONJUNCTIVERULES
GENERATEDISJUNCTIVERULES

C.2 Switcheroo Corridor

The model architectures are listed below:

Model	Architecture
MLP actor MDP	(0): Linear(in=N_STATES, out=4, bias=True) (1): Tanh() (2): Linear(in=4, out=2, bias=True)
MLP actor POMDP	(0): Linear(in=2, out=4, bias=True) (1): Tanh() (2): Linear(in=4, out=2, bias=True)
NDNF-MT actor MDP	(0): SemiSymbolic(in=N_STATES, out=4, type=CONJ.) (1): SemiSymbolicMutexTanh(in=4, out=2, type=DISJ.)
NDNF-MT actor POMDP	(0): SemiSymbolic(in=2, out=4, type=CONJ.) (1): SemiSymbolicMutexTanh(in=4, out=2, type=DISJ.)
Critic MDP	(0): Linear(in=N_STATES, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=1, bias=True)
Critic POMDP	(0): Linear(in=2, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=1, bias=True)

The PPO hyperparameters used for training both the MLP actor and the neural DNF-MT actor are listed below:

Hyperparameter	Value
total_timesteps	1e5
learning_rate	1e−2
num_envs	8
num_steps	64
anneal_lr	True
gamma	0.99
gae_lambda	0.95
num_minibatches	8
update_epochs	4
norm_adv	True
clip_coef	0.3
clip_vloss	True
ent_coef	0.1
vf_coef	1
max_grad_norm	0.5

For the neural DNF-MT actor, the hyperparameters of the auxiliary losses and δ delay scheduling used are listed below:

Hyperparameter Group	Hyperparameter Name	Value
Auxiliary Loss	dis_weight_reg_lambda	0
	conj_tanh_out_reg_lambda	0
	mt_lambda	1e−3
Delta Scheduling	initial_delta	0.1
	delta_decay_delay	30
	delta_decay_steps	5
	delta_decay_rate	1.1

More interpretability examples for Long Corridor-5 (LC-5) and Long Corridor-11 (LC-11) environments are provided in this section.

LC-5. The special state is also at state 1. Listing C.1 is the ASP program for a neural DNF-MT actor in LC-5 MDP, with correctly identifying the special state, state 1.

```
1 action(left) :- in_s_1.
2 action(right) :- not in_s_1.
```

Listing C.1: ASP rules for a neural DNF-MT actor in LC-5 MDP.

Listing C.2 is the ProbLog rules for one neural DNF-MT actor of the runs in LC-5 POMDP. Again, with wall status observations, the actor needs to be 'flexible' when there is no wall on either side, as shown in line 2 in Listing C.2. The probability of going left and right differs from the SC POMDP case, as there is one more state with no walls on either side in LC5.

```
0.139::action(left) ; 0.861::action(right) :- left_wall_present, \+
    right_wall_present.
0.326::action(left) ; 0.674::action(right) :- \+ left_wall_present,
    \+ right_wall_present.
```

Listing C.2: ProbLog rules for a neural DNF-MT actor in LC-5 POMDP.

LC-11. The special states are 5, 6, 7 and 8, while the agent needs to go from state 7 to state 3. The minimal path goes through $[7, 6, 5, 4]$ in order, with action [right, right, right, left].

Listing C.3 is the ASP program for a neural DNF-MT actor in LC-11 MDP. It correctly captures the special states that it will go through in the minimal path in a single conjunction node ('conj_3' in line 3).

```
action(left) :- conj_3.
action(right) :- not conj_3.
conj_3 :- not in_s_5, not in_s_6, not in_s_7.
```

Listing C.3: ASP rules for a neural DNF-MT actor in LC-11 MDP.

Listing C.4 is the ProbLog program for a neural DNF-MT actor in LC-11 POMDP. The left wall will never be present since the agent will never see the state 0. This is reflected in the ProbLog rules, where 'left_wall_present' is never considered. The agent prefers to go right when there is no right wall present. Under action sampling, there is a chance of the agent getting to state 11, where the right wall is present, as reflected in line 2. The probability distribution in state 11 is not 0%-100% but 75.6%-24.4% (line 2). This is because the agent is not guaranteed to get to state 11 consistently, making the training less 'balanced' regarding state-visiting frequency.

```
0.244::action(left) ; 0.756::action(right) :-
    \+ right_wall_present.
0.756::action(left) ; 0.244::action(right) :-
    right_wall_present.
```

Listing C.4: ProbLog rules for a neural DNF-MT actor in LC-11 POMDP.

C.3 Blackjack

The model architectures are listed below:

Model	Architecture
MLP actor	(0): Linear(in=44, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=2, bias=True)
NDNF-MT actor	(0): SemiSymbolic(in=44, out=64, type=CONJ.) (1): SemiSymbolicMutexTanh(in=64, out=2, type=DISJ.)
Critic	(0): Linear(in=44, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=1, bias=True)

The PPO hyperparameters used for training both the MLP actor and the neural DNF-MT actor are listed below:

Hyperparameter	Value
total_timesteps	3e5
learning_rate	1e−3
num_envs	32
num_steps	16
anneal_lr	True
gamma	0.99
gae_lambda	0.95
num_minibatches	16
update_epochs	4
norm_adv	True
clip_coef	0.3
clip_vloss	True
ent_coef	0.1
vf_coef	1
max_grad_norm	0.5

For the neural DNF-MT actor, the hyperparameters of the auxiliary losses and δ delay scheduling used are listed below:

Hyperparameter Group	Hyperparameter Name	Value
Auxiliary Loss	dis_weight_reg_lambda	1e−6
	conj_tanh_out_reg_lambda	0
	mt_lambda	1e−3
Delta Scheduling	initial_delta	0.1
	delta_decay_delay	100
	delta_decay_steps	10
	delta_decay_rate	1.1

The neural DNF-MT actor's ProbLog policy corresponding to Figure 6.6 is shown in Listing C.5. The entire program contains 37 non-probabilistic rules with 'conj_i' as the rule head, and 201 annotated disjunctions with action probabilities.

```
0.873::action(stick) ; 0.127::action(hit) :- \+conj_3, \+conj_4, \+
    conj_5, \+conj_6, \+conj_7, \+conj_9, \+conj_10, \+conj_11, \+
    conj_12, \+conj_14, \+conj_15, \+conj_16, \+conj_19, \+conj_21,
    \+conj_22, \+conj_25, \+conj_29, \+conj_32, \+conj_34, \+conj_37,
     \+conj_40, \+conj_42, \+conj_44, \+conj_45, \+conj_47, \+conj_48
    , \+conj_49, \+conj_50, \+conj_51, \+conj_52, \+conj_53, \+
    conj_54, \+conj_55, \+conj_56, \+conj_57, \+conj_59, \+conj_62.
0.907::action(stick) ; 0.093::action(hit) :- \+conj_3, \+conj_4, \+
    conj_5, \+conj_6, \+conj_7, \+conj_9, \+conj_10, \+conj_11, \+
    conj_12, \+conj_14, \+conj_15, \+conj_16, \+conj_19, \+conj_21,
    \+conj_22, \+conj_25, \+conj_29, \+conj_32, \+conj_34, \+conj_37,
     \+conj_40, \+conj_42, \+conj_44, \+conj_45, \+conj_47, \+conj_48
    , \+conj_49, \+conj_50, \+conj_51, \+conj_52, \+conj_53, \+
    conj_54, conj_55, \+conj_56, \+conj_57, \+conj_59, \+conj_62.
0.900::action(stick) ; 0.100::action(hit) :- \+conj_3, \+conj_4, \+
    conj_5, \+conj_6, \+conj_7, \+conj_9, \+conj_10, \+conj_11, \+
    conj_12, \+conj_14, \+conj_15, \+conj_16, \+conj_19, \+conj_21,
    \+conj_22, \+conj_25, \+conj_29, \+conj_32, conj_34, \+conj_37,
    \+conj_40, \+conj_42, \+conj_44, \+conj_45, \+conj_47, \+conj_48,
     \+conj_49, \+conj_50, \+conj_51, \+conj_52, conj_53, \+conj_54,
    \+conj_55, \+conj_56, \+conj_57, \+conj_59, \+conj_62.
0.927::action(stick) ; 0.073::action(hit) :- \+conj_3, \+conj_4, \+
    conj_5, \+conj_6, \+conj_7, \+conj_9, \+conj_10, \+conj_11, \+
    conj_12, \+conj_14, \+conj_15, \+conj_16, \+conj_19, \+conj_21,
    \+conj_22, \+conj_25, \+conj_29, \+conj_32, conj_34, \+conj_37,
    \+conj_40, \+conj_42, \+conj_44, \+conj_45, \+conj_47, \+conj_48,
     \+conj_49, \+conj_50, \+conj_51, \+conj_52, conj_53, \+conj_54,
    conj_55, \+conj_56, \+conj_57, \+conj_59, \+conj_62.
0.983::action(stick) ; 0.017::action(hit) :- \+conj_3, \+conj_4, \+
```

```
    conj_5, \+conj_6, \+conj_7, \+conj_9, \+conj_10, \+conj_11, \+
    conj_12, \+conj_14, \+conj_15, \+conj_16, \+conj_19, \+conj_21,
    \+conj_22, \+conj_25, \+conj_29, \+conj_32, conj_34, \+conj_37,
    \+conj_40, \+conj_42, conj_44, \+conj_45, \+conj_47, \+conj_48,
    \+conj_49, \+conj_50, \+conj_51, \+conj_52, conj_53, \+conj_54,
    conj_55, \+conj_56, \+conj_57, \+conj_59, \+conj_62.
...
conj_3 :-
    hand(18), \+dealer(9), \+dealer(10), \+usable_ace.
conj_4 :-
    \+hand(17), \+hand(18), \+hand(19), \+hand(20), \+hand(21), \+
    hand(22), \+hand(28).
conj_5 :-
    hand(20), \+dealer(1), \+dealer(9), \+dealer(10), \+usable_ace.
conj_6 :-
    hand(12), \+dealer(5), usable_ace.
conj_7 :-
    hand(10), \+dealer(2), \+dealer(3), \+dealer(4), \+dealer(6).
...
```

Listing C.5: ProbLog rules for the neural DNF-MT actor as shown in Figure 6.6, showing the first five annotated disjunctions and the first five conjunction rules.

C.4 Taxi

The model architectures are listed below:

Model	Architecture
MLP actor	(0): Linear(in=500, out=256, bias=True) (1): Tanh() (2): Linear(in=256, out=6, bias=True)
Critic	(0): Linear(in=500, out=256, bias=True) (1): ReLU() (2): Linear(in=256, out=256, bias=True) (3): ReLU() (4): Linear(in=256, out=1, bias=True)
MLP oracle (for distillation)	(0): Linear(in=500, out=256, bias=False) (1): Tanh() (2): Linear(in=256, out=6, bias=False)
NDNF-MT actor (distilled)	(0): SemiSymbolic(in=500, out=64, type=CONJ.) (1): SemiSymbolicMutexTanh(in=64, out=6, type=DISJ.)

The PPO hyperparameters used for training the MLP actor are listed below:

Hyperparameter	Value
total_timesteps	3e6
learning_rate_actor	2e−4
learning_rate_critic	2e−3
num_envs	64
num_steps	2048
anneal_lr	True
gamma	0.999
gae_lambda	0.946
num_minibatches	128
update_epochs	8
norm_adv	True
clip_coef	0.2
clip_vloss	True
ent_coef	0.003
vf_coef	0.5
max_grad_norm	0.5

The distillation hyperparameters used for training neural DNF-MT actors are listed below:

Hyperparameter Group	Hyperparameter Name	Value
Distilaltion	batch_size	32
	epoch	5000
	learning_rate	1e−4
Auxiliary Loss	dis_weight_reg_lambda	1e−4
	conj_tanh_out_reg_lambda	1e−5
	mt_lambda	1e−4
Delta Scheduling	initial_delta	0.1
	delta_decay_delay	1000
	delta_decay_steps	100
	delta_decay_rate	1.1

An example of a neural DNF-MT actor's ProbLog rules is provided in Listing C.6.

```
0.595::action(down) ; 0.000::action(up) ; 0.000::action(right) ;
    0.405::action(left) ; 0.000::action(pickup) ; 0.000::action(
    dropoff) :- \+conj_0, \+conj_1, \+conj_3, \+conj_4, conj_5,
    conj_6, \+conj_7, \+conj_8, \+conj_9, \+conj_11, \+conj_12, \+
    conj_13, \+conj_14, conj_15, \+conj_16, \+conj_18, \+conj_19,
    conj_20, conj_21, \+conj_22, conj_23, conj_24, \+conj_25, \+
```

```
    conj_26, conj_27, \+conj_28, \+conj_29, conj_30, \+conj_31, \+
    conj_33, \+conj_34, \+conj_35, \+conj_37, \+conj_38, \+conj_39,
    conj_40, \+conj_41, \+conj_42, \+conj_43, \+conj_44, \+conj_45,
    \+conj_46, \+conj_47, \+conj_48, \+conj_50, \+conj_51, \+conj_52,
     \+conj_55, \+conj_56, conj_57, \+conj_58, conj_60, \+conj_62,
    conj_63.
2 0.602::action(down) ; 0.000::action(up) ; 0.000::action(right) ;
    0.398::action(left) ; 0.000::action(pickup) ; 0.000::action(
    dropoff) :- \+conj_0, \+conj_1, \+conj_3, \+conj_4, conj_5,
    conj_6, \+conj_7, \+conj_8, \+conj_9, \+conj_11, conj_12, \+
    conj_13, \+conj_14, conj_15, \+conj_16, \+conj_18, \+conj_19,
    conj_20, conj_21, \+conj_22, conj_23, conj_24, \+conj_25, \+
    conj_26, conj_27, \+conj_28, \+conj_29, conj_30, \+conj_31, \+
    conj_33, \+conj_34, \+conj_35, \+conj_37, \+conj_38, \+conj_39,
    conj_40, \+conj_41, \+conj_42, \+conj_43, \+conj_44, \+conj_45,
    \+conj_46, \+conj_47, \+conj_48, \+conj_50, \+conj_51, \+conj_52,
     \+conj_55, \+conj_56, conj_57, \+conj_58, conj_60, \+conj_62,
    conj_63.
3 0.963::action(down) ; 0.000::action(up) ; 0.000::action(right) ;
    0.036::action(left) ; 0.000::action(pickup) ; 0.001::action(
    dropoff) :- \+conj_0, \+conj_1, \+conj_3, \+conj_4, conj_5,
    conj_6, \+conj_7, \+conj_8, \+conj_9, \+conj_11, conj_12, \+
    conj_13, \+conj_14, conj_15, \+conj_16, \+conj_18, \+conj_19,
    conj_20, conj_21, \+conj_22, conj_23, conj_24, \+conj_25, \+
    conj_26, conj_27, \+conj_28, \+conj_29, conj_30, \+conj_31, \+
    conj_33, \+conj_34, \+conj_35, \+conj_37, \+conj_38, \+conj_39,
    conj_40, \+conj_41, \+conj_42, \+conj_43, \+conj_44, \+conj_45,
    \+conj_46, \+conj_47, conj_48, \+conj_50, \+conj_51, \+conj_52,
    \+conj_55, \+conj_56, \+conj_57, \+conj_58, conj_60, \+conj_62,
    conj_63.
4 0.000::action(down) ; 0.000::action(up) ; 0.000::action(right) ;
    0.000::action(left) ; 1.000::action(pickup) ; 0.000::action(
    dropoff) :- conj_0, conj_1, conj_3, conj_4, conj_5, conj_6, \+
    conj_7, \+conj_8, \+conj_9, conj_11, conj_12, \+conj_13, \+
    conj_14, conj_15, \+conj_16, \+conj_18, conj_19, conj_20, conj_21
    , \+conj_22, conj_23, conj_24, \+conj_25, \+conj_26, conj_27,
    conj_28, \+conj_29, \+conj_30, \+conj_31, \+conj_33, \+conj_34,
    \+conj_35, \+conj_37, \+conj_38, conj_39, \+conj_40, \+conj_41,
    \+conj_42, \+conj_43, conj_44, \+conj_45, \+conj_46, \+conj_47,
    conj_48, \+conj_50, \+conj_51, conj_52, \+conj_55, \+conj_56,
    conj_57, \+conj_58, conj_60, \+conj_62, conj_63.
5 0.000::action(down) ; 0.000::action(up) ; 0.000::action(right) ;
    0.000::action(left) ; 0.000::action(pickup) ; 1.000::action(
```

```
    dropoff) :- \+conj_0, conj_1, conj_3, \+conj_4, \+conj_5, conj_6,
     \+conj_7, \+conj_8, \+conj_9, conj_11, conj_12, \+conj_13, \+
    conj_14, \+conj_15, \+conj_16, \+conj_18, \+conj_19, conj_20, \+
    conj_21, \+conj_22, conj_23, \+conj_24, \+conj_25, \+conj_26,
    conj_27, \+conj_28, conj_29, \+conj_30, \+conj_31, \+conj_33, \+
    conj_34, \+conj_35, \+conj_37, \+conj_38, conj_39, conj_40, \+
    conj_41, \+conj_42, \+conj_43, conj_44, \+conj_45, \+conj_46, \+
    conj_47, conj_48, \+conj_50, \+conj_51, \+conj_52, \+conj_55, \+
    conj_56, \+conj_57, \+conj_58, conj_60, \+conj_62, conj_63.
...
conj_0 :- \+state(4), \+state(9), \+state(11), \+state(14), \+state
    (16), \+state(24), \+state(26), \+state(28), \+state(33), \+state
    (52), \+state(54), \+state(59), \+state(61), \+state(68), \+state
    (72), \+state(76), \+state(78), \+state(79), \+state(81), \+state
    (82), \+state(89), \+state(94), \+state(96), \+state(97), \+state
    (104), \+state(107), \+state(112), \+state(113), \+state(114), \+
    state(128), \+state(129), \+state(131), \+state(134), \+state
    (139), \+state(152), \+state(159), \+state(162), \+state(171), \+
    state(176), \+state(178), \+state(182), \+state(189), \+state
    (191), \+state(193), \+state(194), \+state(279), \+state(311), \+
    state(379), \+state(394), \+state(443), \+state(479), \+state
    (489).
conj_1 :- \+state(14), \+state(21), \+state(28), \+state(61), \+
    state(62), \+state(63), \+state(68), \+state(69), \+state(76), \+
    state(78), \+state(81), \+state(82), \+state(88), \+state(96), \+
    state(101), \+state(121), \+state(128), \+state(131), \+state
    (136), \+state(138), \+state(144), \+state(162), \+state(163), \+
    state(166), \+state(168), \+state(176), \+state(177), \+state
    (178), \+state(182), \+state(186), \+state(187), \+state(188), \+
    state(189), \+state(191), \+state(193), \+state(196), \+state
    (238), \+state(246), \+state(256), \+state(266), \+state(267), \+
    state(276), \+state(278), \+state(281), \+state(286), \+state
    (298), \+state(303), \+state(327), \+state(331), \+state(332), \+
    state(333), \+state(339), \+state(341), \+state(342), \+state
    (343), \+state(344), \+state(348), \+state(349), \+state(351), \+
    state(352), \+state(353), \+state(354), \+state(359), \+state
    (363), \+state(366), \+state(367), \+state(382), \+state(383), \+
    state(384), \+state(388), \+state(391), \+state(394), \+state
    (396), \+state(398), \+state(401), \+state(402), \+state(404), \+
    state(406), \+state(412), \+state(413), \+state(414), \+state
    (423), \+state(427), \+state(428), \+state(431), \+state(434), \+
    state(437), \+state(438), \+state(439), \+state(441), \+state
    (443), \+state(448), \+state(449), \+state(451), \+state(452), \+
```

```
    state(454), \+state(457), \+state(458), \+state(459), \+state
    (462), \+state(464), \+state(466), \+state(468), \+state(469), \+
    state(481), \+state(483), \+state(484), \+state(488), \+state
    (489), \+state(491), \+state(496), \+state(498).
conj_3 :- \+state(6), \+state(11), \+state(14), \+state(17), \+state
    (24), \+state(27), \+state(28), \+state(32), \+state(33), \+state
    (38), \+state(51), \+state(52), \+state(54), \+state(58), \+state
    (59), \+state(61), \+state(68), \+state(72), \+state(76), \+state
    (78), \+state(79), \+state(81), \+state(88), \+state(91), \+state
    (93), \+state(94), \+state(96), \+state(104), \+state(107), \+
    state(114), \+state(117), \+state(128), \+state(129), \+state
    (131), \+state(132), \+state(133), \+state(134), \+state(139), \+
    state(152), \+state(156), \+state(159), \+state(162), \+state
    (169), \+state(171), \+state(173), \+state(176), \+state(178), \+
    state(182), \+state(183), \+state(189), \+state(191), \+state
    (192), \+state(193), \+state(194), \+state(199), \+state(211), \+
    state(293), \+state(294), \+state(308), \+state(309), \+state
    (311), \+state(318), \+state(379), \+state(392), \+state(393), \+
    state(394), \+state(414), \+state(436), \+state(439), \+state
    (452).
conj_4 :- \+state(4), \+state(6), \+state(8), \+state(9), \+state
    (12),
         \+state(14), \+state(16), \+state(18), \+state(19), \+state
    (24),
         \+state(26), \+state(27), \+state(29), \+state(32), \+state
    (33),
         \+state(34), \+state(37), \+state(38), \+state(39), \+state
    (42),
         \+state(43), \+state(48), \+state(51), \+state(56), \+state
    (59),
         \+state(61), \+state(68), \+state(71), \+state(73), \+state
    (74),
         \+state(76), \+state(79), \+state(82), \+state(83), \+state
    (88),
         \+state(91), \+state(92), \+state(93), \+state(94), \+state
    (97),
         \+state(99), \+state(101), \+state(104), \+state(108), \+
    state(109),
         \+state(111), \+state(112), \+state(113), \+state(117), \+
    state(119),
         \+state(121), \+state(126), \+state(127), \+state(129), \+
    state(131),
         \+state(132), \+state(133), \+state(134), \+state(136), \+
```

```
    state(137),
22        \+state(139), \+state(141), \+state(143), \+state(146), \+
    state(148),
23        \+state(151), \+state(158), \+state(162), \+state(166), \+
    state(169),
24        \+state(171), \+state(172), \+state(173), \+state(174), \+
    state(176),
25        \+state(177), \+state(178), \+state(179), \+state(183), \+
    state(184),
26        \+state(186), \+state(187), \+state(189), \+state(191), \+
    state(192),
27        \+state(193), \+state(194), \+state(201), \+state(203), \+
    state(208),
28        \+state(211), \+state(218), \+state(222), \+state(266), \+
    state(267),
29        \+state(272), \+state(273), \+state(274), \+state(277), \+
    state(284),
30        \+state(286), \+state(293), \+state(294), \+state(297), \+
    state(299),
31        \+state(301), \+state(302), \+state(303), \+state(304), \+
    state(306),
32        \+state(307), \+state(309), \+state(311), \+state(312), \+
    state(313),
33        \+state(314), \+state(321), \+state(322), \+state(323), \+
    state(328),
34        \+state(332), \+state(334), \+state(338), \+state(339), \+
    state(341),
35        \+state(343), \+state(344), \+state(346), \+state(347), \+
    state(351),
36        \+state(352), \+state(353), \+state(361), \+state(362), \+
    state(368),
37        \+state(369), \+state(371), \+state(372), \+state(373), \+
    state(382),
38        \+state(383), \+state(384), \+state(386), \+state(389), \+
    state(391),
39        \+state(392), \+state(393), \+state(394), \+state(396), \+
    state(397),
40        \+state(401), \+state(402), \+state(403), \+state(404), \+
    state(406),
41        \+state(407), \+state(414), \+state(418), \+state(419), \+
    state(421),
42        \+state(422), \+state(427), \+state(429), \+state(433), \+
    state(436),
```

```
43          \+state(438), \+state(442), \+state(444), \+state(446), \+
    state(447),
44          \+state(449), \+state(451), \+state(453), \+state(454), \+
    state(456),
45          \+state(457), \+state(458), \+state(459), \+state(461), \+
    state(462),
46          \+state(463), \+state(467), \+state(468), \+state(471), \+
    state(477),
47          \+state(482), \+state(486), \+state(487), \+state(489), \+
    state(491),
48          \+state(496), \+state(497), \+state(498).
49 conj_5 :- \+state(16), \+state(97), \+state(123), \+state(136), \+
    state(479).
50 ...
```

Listing C.6: ProbLog rules for a distilled neural DNF-MT actor in the Taxi environment, showing five annotated disjunctions and first five conjunction rules.

C.5 Door Corridor

The model architectures are listed below:

Model	Architecture
Feature Encoder	(0): Conv2d(2, 4, kernel_size=(1, 1), stride=(1, 1)) (1): Tanh() (2): Linear(in=36, out=16, bias=True)
MLP actor	(0): Linear(in=16, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=4, bias=True)
NDNF-MT Actor	(0): SemiSymbolic(in=16, out=12, type=CONJ.) (1): SemiSymbolicMutexTanh(in=12 out=4, type=DISJ.)
Critic	(0): Linear(in=16, out=64, bias=True) (1): Tanh() (2): Linear(in=64, out=1, bias=True)

The PPO hyperparameters used for training both the MLP actor and the neural DNF-MT actor are listed below:

Hyperparameter	Value
total_timesteps	3e5
learning_rate	1e−2
num_envs	8
num_steps	64
anneal_lr	True
gamma	0.99
gae_lambda	0.95
num_minibatches	8
update_epochs	4
norm_adv	True
clip_coef	0.3
clip_vloss	True
ent_coef	0.1
vf_coef	1
max_grad_norm	0.5

For the neural DNF-MT actor, the hyperparameters of the auxiliary losses and δ delay scheduling used are listed below:

Hyperparameter Group	Hyperparameter Name	Value
Auxiliary Loss	dis_weight_reg_lambda	0
	conj_tanh_out_reg_lambda	0
	mt_lambda	1e−3
	embedding_reg_lambda	3e−15
Delta Scheduling	initial_delta	0.1
	delta_decay_delay	50
	delta_decay_steps	10
	delta_decay_rate	1.1

The code repo provides a notebook on policy intervention in DC-T and DC-OT. The notebook's path is `notebooks/Door Corridor PPO NDNF-MT-6731 Intervention.ipynb` (*link to notebook*).

Appendix D

Proofs of Disentanglement Method

Proof of Proposition 7.3.2. This proof shows that $\mathcal{W}$ constructed under Proposition 7.3.2 satisfies the constraints in Equation (7.8): the positive coverage is straightforward and the negative coverage is proved by contradiction.

Case 1: Positive Coverage – All Positive Examples are Covered (Constraint in Equation (7.9))

For any positive example $\mathbf{x}_i \in \mathcal{X}^+$, take $\tilde{\mathbf{w}}_i$ associated to $\mathbf{x}_i$. By the definition of $\tilde{\mathbf{w}}_i$ in Equation (7.11), the following holds:

$$\begin{aligned} f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_i) &= \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{i,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}|\right) \\ &= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} = \text{sign}(w_j)\right) + \left(6 - 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} = \text{sign}(w_j)\right)\right) \\ &= 6 \end{aligned}$$

Thus, the constraint in Equation (7.9) is satisfied and the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^+|}\}$ will cover all positive examples $\mathbf{x}_i \in \mathcal{X}^+$.

Case 2: Negative Coverage – No Negative Examples are Covered (Constraint in Equation (7.10))

The proof shows by contradiction that the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^+|}\}$ constructed under Proposition 7.3.2 does not cover any of the negative examples.

Assume that a negative example $\mathbf{x}_m \in \mathcal{X}^-$ is covered by $\tilde{\mathbf{w}}_i$, which is associated to a positive example $\mathbf{x}_i \in \mathcal{X}^+$. Then the raw output of $\tilde{\mathbf{w}}_i$ over $\mathbf{x}_m$ should be 6 to cover $\mathbf{x}_m$:

$$\begin{aligned} & f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_m) = 6 \\ & \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{m,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}|\right) = 6 \\ \Rightarrow & \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{m,j} = \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}| \end{aligned} \tag{D.1}$$

Substitute the definition of $\tilde{w}_{i,j}$ in Equation (7.11) into Equation (D.1), and the following holds:

$$\sum_{j \in \mathcal{J}} x_{i,j} \cdot \mathbb{1}\left(x_{i,j} = \text{sign}(w_j)\right) \cdot x_{m,j} = \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} = \text{sign}(w_j)\right) \tag{D.2}$$

When $x_{i,j} \neq \text{sign}(w_j)$, $\mathbb{1}\left(x_{i,j} = \text{sign}(w_j)\right) = 0$, and will not contribute to the summation term in either the L.H.S. or the R.H.S. of Equation (D.2). Thus, for Equation (D.2) to hold, this has to hold:

$$\forall j \in \mathcal{J}.\ \left[x_{i,j} = \text{sign}(w_j) \implies x_{m,j} = x_{i,j}\right] \tag{D.3}$$

From Equation (D.3), $\{j \in \mathcal{J} | x_{i,j} = \text{sign}(w_j)\} \subset \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j)\}$. So $\{j \in \mathcal{J} | x_{i,j} \neq \text{sign}(w_j)\} \supset \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j)\}$. The subset cannot be equal because $\mathbf{x}_i \neq \mathbf{x}_m$. The following set relations hold:

$$\begin{aligned} &\{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j)\} \\ &= \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \cup \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \end{aligned} \tag{D.4}$$

$$\{j \in \mathcal{J} | x_{i,j} = \text{sign}(w_j)\} = \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \tag{D.5}$$

$$\{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j)\} = \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \tag{D.6}$$

$$\begin{aligned} &\{j \in \mathcal{J} | x_{i,j} \neq \text{sign}(w_j)\} \\ &= \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \cup \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \end{aligned} \tag{D.7}$$

$$\{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \neq \emptyset \tag{D.8}$$

Figure D.1 helps to demonstrate the relations between the sets.

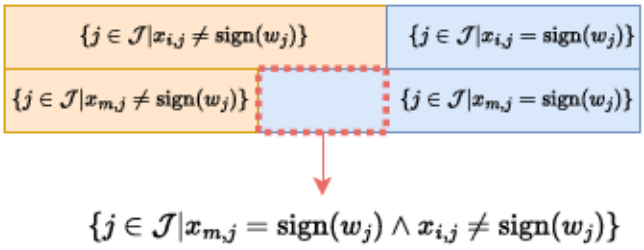


Figure D.1: Relations between the sign-matching subsets of $\mathbf{x}_i$ and $\mathbf{x}_m$.

Using the definition of the semi-symbolic node in Equation (4.1), the raw output of the original conjunctive node with weight $\mathbf{w}$ over the negative example $\mathbf{x}_m$ is computed as:

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}_m) &= \sum_{j \in \mathcal{J}} w_j x_{m,j} + \beta \\ &= \sum_{j \in \mathcal{J}: x_{m,j} = \text{sign}(w_j)} |w_j| - \sum_{j \in \mathcal{J}: x_{m,j} \neq \text{sign}(w_j)} |w_j| + \beta \end{aligned}$$

where the bias term β is not expanded. Using the set relations in Equation (D.4) and Equation (D.6),

$f_{\mathbf{w}}(\mathbf{x}_m)$ can be rewritten as:

$$f_{\mathbf{w}}(\mathbf{x}_m) = \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}=\text{sign}(w_j)}} |w_j| + \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| - \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}\neq\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| + \beta \tag{D.9}$$

Because the relation in Equation (D.8) holds, the second term $\sum_{j\in\mathcal{J}:x_{m,j}=\text{sign}(w_j),x_{i,j}\neq\text{sign}(w_j)} |w_j|$ in the R.H.S. of Equation (D.9) is guaranteed to be greater than 0. Thus, the following inequality holds:

$$f_{\mathbf{w}}(\mathbf{x}_m) > f_{\mathbf{w}}(\mathbf{x}_m) - 2 \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| \tag{D.10}$$

where the R.H.S. of the inequality is computed as:

$$\begin{aligned} & f_{\mathbf{w}}(\mathbf{x}_m) - 2 \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| \\ = & \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}=\text{sign}(w_j)}} |w_j| - \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| - \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}\neq\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| + \beta \end{aligned} \tag{D.11}$$

Equation (D.11) can be transformed as:

$$\begin{aligned} & \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}=\text{sign}(w_j)}} |w_j| - \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| - \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}\neq\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| + \beta \\ = & \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}=\text{sign}(w_j)}} |w_j| - \left(\sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| + \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}\neq\text{sign}(w_j),\\ x_{i,j}\neq\text{sign}(w_j)}} |w_j| \right) + \beta \\ = & \sum_{\substack{j\in\mathcal{J}:\\ x_{m,j}=\text{sign}(w_j),\\ x_{i,j}=\text{sign}(w_j)}} |w_j| - \sum_{j\in\mathcal{J}:x_{i,j}\neq\text{sign}(w_j)} |w_j| + \beta && \text{by Eq. (D.7)} \\ = & \sum_{x_{i,j}=\text{sign}(w_j)} |w_j| - \sum_{j\in\mathcal{J}:x_{i,j}\neq\text{sign}(w_j)} |w_j| + \beta && \text{by Eq. (D.5)} \\ = & f_{\mathbf{w}}(\mathbf{x}_t) && \text{by node def. in Eq. (4.1)} \\ > & 0 && \mathbf{x}_t \in \mathcal{X}^+ \end{aligned}$$

This gives $f_{\mathbf{w}}(\mathbf{x}_m) > f_{\mathbf{w}}(\mathbf{x}_t) > 0$, but this contradicts the assumption that $\mathbf{x}_m \in \mathcal{X}^-$ such that $f_{\mathbf{w}}(\mathbf{x}_m) \leq 0$. This proves that no negative examples will be covered by the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^+|}\}$ constructed under Equation (7.11).

This shows that the split weight set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^+|}\}$ constructed under Proposition 7.3.2 will cover all positive examples and none of the negative examples.

■

Proof of Proposition 7.3.3. This proof shows that $\mathcal{W}$ constructed under Proposition 7.3.3 satisfies the constraints in Equation (7.13): the negative coverage is straightforward and the positive coverage is proved by contradiction.

Case 1: Negative Coverage – All Negative Examples are Covered (Constraint in Equation (7.14))

For any negative example $\mathbf{x}_i \in \mathcal{X}^-$, take $\tilde{\mathbf{w}}_i$ associated to $\mathbf{x}_i$. By the definition of $\tilde{\mathbf{w}}_i$ in Equation (7.16), the following holds:

$$\begin{aligned} f_{\tilde{\mathbf{w}}_i}(\mathbf{x}_i) &= \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{i,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}|\right) \\ &= 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} \neq \text{sign}(w_j)\right) + \left(6 - 6 \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} \neq \text{sign}(w_j)\right)\right) \\ &= 6 \end{aligned}$$

Thus, the constraint in Equation (7.14) is satisfied and the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^-|}\}$ will cover all negative examples $\mathbf{x}_i \in \mathcal{X}^-$.

Case 2: Positive Coverage – No Positive Examples are Covered (Constraint in Equation (7.15))

The proof shows by contradiction that the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^-|}\}$ constructed under Proposition 7.3.3 does not cover any of the positive examples.

Assume that a positive example $\mathbf{x}_m \in \mathcal{X}^+$ is covered by $\tilde{\mathbf{w}}_i$, which is associated to a negative example $\mathbf{x}_i \in \mathcal{X}^-$. Then the raw output of $\tilde{\mathbf{w}}_i$ over $\mathbf{x}_m$ should be 6 to cover $\mathbf{x}_m$:

$$\begin{aligned} & f_{\tilde{\mathbf{w}}}(\mathbf{x}_m) = 6 \\ & \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{m,j} + \left(6 - \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}|\right) = 6 \\ \Rightarrow & \sum_{j \in \mathcal{J}} \tilde{w}_{i,j} x_{m,j} = \sum_{j \in \mathcal{J}} |\tilde{w}_{i,j}| \end{aligned} \tag{D.12}$$

Substitute the definition of $\tilde{w}_{i,j}$ in Equation (7.16) into Equation (D.12), and the following holds:

$$\sum_{j \in \mathcal{J}} x_{i,j} \cdot \mathbb{1}\left(x_{i,j} \neq \text{sign}(w_j)\right) \cdot x_{m,j} = \sum_{j \in \mathcal{J}} \mathbb{1}\left(x_{i,j} \neq \text{sign}(w_j)\right) \tag{D.13}$$

When $x_{i,j} = \text{sign}(w_j)$, $\mathbb{1}\left(x_{i,j} \neq \text{sign}(w_j)\right) = 0$, and will not contribute to the summation term in either the L.H.S. or the R.H.S. of Equation (D.13). Thus, for Equation (D.13) to hold, this has to hold:

$$\forall j \in \mathcal{J}.\ \left[x_{i,j} \neq \text{sign}(w_j) \implies x_{m,j} = x_{i,j}\right] \tag{D.14}$$

From Equation (D.14), $\{j \in \mathcal{J} | x_{i,j} \neq \text{sign}(w_j)\} \subset \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j)\}$. So $\{j \in \mathcal{J} | x_{i,j} = \text{sign}(w_j)\} \supset \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j)\}$. The subset cannot be equal because $\mathbf{x}_i \neq \mathbf{x}_m$. The following set relations hold:

$$\begin{aligned} & \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j)\} \\ = & \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \cup \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \end{aligned} \tag{D.15}$$

$$\{j \in \mathcal{J} | x_{i,j} \neq \text{sign}(w_j)\} = \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} \neq \text{sign}(w_j)\} \tag{D.16}$$

$$\{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j)\} = \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \tag{D.17}$$

$$\begin{aligned} &\{j \in \mathcal{J} | x_{i,j} = \text{sign}(w_j)\} \\ &= \{j \in \mathcal{J} | x_{m,j} = \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \cup \{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \end{aligned} \tag{D.18}$$

$$\{j \in \mathcal{J} | x_{m,j} \neq \text{sign}(w_j) \wedge x_{i,j} = \text{sign}(w_j)\} \neq \emptyset \tag{D.19}$$

Figure D.2 helps to demonstrate the relations between the sets.

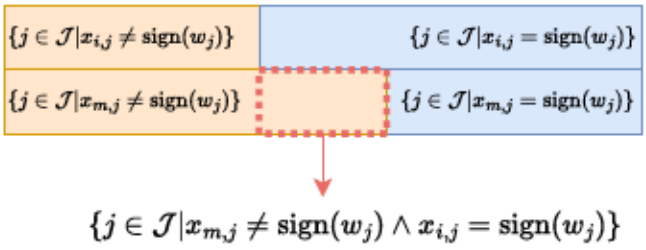


Figure D.2: Relations between the sign-mismatching subsets of $\mathbf{x}_i$ and $\mathbf{x}_m$.

Using the definition of the semi-symbolic node in Equation (4.1), the raw output of the original conjunctive node with weight $\mathbf{w}$ over the positive example $\mathbf{x}_m$ is computed as:

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}_m) &= \sum_{j \in \mathcal{J}} w_j x_{m,j} + \beta \\ &= \sum_{j \in \mathcal{J}: x_{m,j} = \text{sign}(w_j)} |w_j| - \sum_{j \in \mathcal{J}: x_{m,j} \neq \text{sign}(w_j)} |w_j| + \beta \end{aligned}$$

where the bias term β is not expanded. Using the set relations in Equation (D.15) and Equation (D.17), $f_{\mathbf{w}}(\mathbf{x}_m)$ can be rewritten as:

$$f_{\mathbf{w}}(\mathbf{x}_m) = \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} = \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| - \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| - \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} \neq \text{sign}(w_j)}} |w_j| + \beta \tag{D.20}$$

Because the relation in Equation (D.19) holds, the second term $\sum_{j \in \mathcal{J}: x_{m,j} \neq \text{sign}(w_j), x_{i,j} = \text{sign}(w_j)} |w_j|$ in the R.H.S. of Equation (D.20) is guaranteed to be greater than 0. Thus, the following inequality holds:

$$f_{\mathbf{w}}(\mathbf{x}_m) < f_{\mathbf{w}}(\mathbf{x}_m) + 2 \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| \tag{D.21}$$

where the R.H.S. of the inequality is computed as:

$$\begin{aligned} & f_{\mathbf{w}}(\mathbf{x}_m) \quad + 2 \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| \\ = & \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} = \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| + \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| - \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} \neq \text{sign}(w_j)}} |w_j| + \beta \end{aligned} \tag{D.22}$$

Equation (D.22) can be transformed as:

$$\begin{aligned} & \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} = \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| + \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| - \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} \neq \text{sign}(w_j)}} |w_j| + \beta \\ = & \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} = \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| + \sum_{\substack{j \in \mathcal{J}: \\ x_{m,j} \neq \text{sign}(w_j), \\ x_{i,j} = \text{sign}(w_j)}} |w_j| - \sum_{j \in \mathcal{J}: x_{i,j} \neq \text{sign}(w_j)} |w_j| + \beta && \text{by Eq. (D.16)} \\ = & \sum_{j \in \mathcal{J}: x_{i,j} = \text{sign}(w_j)} |w_j| - \sum_{j \in \mathcal{J}: x_{i,j} \neq \text{sign}(w_j)} |w_j| + \beta && \text{by Eq. (D.18)} \\ = & f_{\mathbf{w}}(\mathbf{x}_i) && \text{by node def. in Eq. (4.1)} \\ \leq & 0 && \mathbf{x}_i \in \mathcal{X}^- \end{aligned}$$

This gives $f_{\mathbf{w}}(\mathbf{x}_m) < f_{\mathbf{w}}(\mathbf{x}_i) \leq 0$, but this contradicts the assumption that $\mathbf{x}_m \in \mathcal{X}^+$ such that $f_{\mathbf{w}}(\mathbf{x}_m) > 0$. This proves that no positive examples will be covered by the set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^-|}\}$ constructed under Equation (7.16).

This shows that the split weight set $\{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_{|\mathcal{X}^-|}\}$ constructed under Proposition 7.3.3 will cover all negative examples and none of the positive examples.

■

Appendix E

Code Repositories

This appendix provides a comprehensive list of the code repositories that accompany this thesis. All repositories are publicly available on GitHub.

Core Library of Neural DNF-based Models. The main neural DNF-based models library is available at `https://github.com/kittykg/neural-dnf`. This repository contains the core implementation of the neural DNF-based models framework, including the semi-symbolic layer implementation, the family of neural DNF-based models (neural DNF, neural DNF-EO, and neural DNF-MT), delta scheduling methods for training, and the post-training processes for rule extraction. The library provides the foundational components used across all experimental work in this thesis.

CUB Dataset/subsets Experiments. The experiments on the CUB-200-2011 dataset and its subsets presented in Chapter 5 are available at `https://github.com/kittykg/neural-dnf-cub`. This repository contains the implementation and evaluation scripts for the multi-class classification experiments using the neural DNF and neural DNF-EO models.

TMC Dataset/subsets Experiments. The experiments on the TMC2007-500 dataset and its subsets presented in Chapter 5 are available at `https://github.com/kittykg/neural-dnf-tmc`. This repository contains the implementation and evaluation scripts for the multi-label classification experiments using the neural DNF model.

Reinforcement Learning Experiments. The RL experiments presented in Chapter 6 are available at `https://github.com/kittykg/neural-dnf-mt-policy-learning`. This repository contains the integration of the neural DNF-MT model with actor-critic algorithms, the translation between neural DNF-MT models and probabilistic logic programs, and the evaluation scripts for the RL environments.

Disentanglement Experiments. The experiments presented in Chapter 7 are available at `https://github.com/kittykg/disentangling-ndnf-classification`. This repository contains the implementation of the disentanglement method in the post-training process for neural DNF-based models, the threshold learning method for predicate invention, and the improved delta scheduling

method. The repository also includes evaluation scripts for the classification tasks used to demonstrate the effectiveness of the disentanglement approach.