

Department of Computing
Imperial College London

Systematic Generalisation in Deep Reinforcement Learning

Borja Gonzalez Leon

Submitted March 29, 2024, in partial fulfilment of the
requirements for the degree of Doctor of Philosophy

Abstract

One of the main goals in artificial intelligence (AI) is to build autonomous agents that correctly execute complex instructions given by a human user in a wide variety of settings. A promising line of work towards such objective is known as *grounded language learning*, in which autonomous agents are trained to not only parse language-based instructions, but to carry out those instructions in an environment, either physical or digital. A dominant paradigm for this line of work is deep reinforcement learning (DRL), an approach that blends the fields of reinforcement learning and artificial neural networks (NNs).

However, despite its promise, DRL solutions remain fragile. A major challenge in creating robust and efficient agents for grounded language learning is achieving systematic generalisation, the ability to comprehend and to generate new combinations of familiar elements. Systematic generalisation in a grounded setting is a recurrent research topic, and even the most advanced contemporary AI systems, large language models, struggle to achieve it.

In this thesis, we first present theoretical frameworks for combining temporal logic, a formal language suited for grounded language learning, and DRL. Then, we present two empirical contributions towards systematic generalisation in DRL. The first one concerns the impact of visual encoders when learning abstract operators such as negation in a systematic fashion. The second contribution are novel neural-network architectures that enable more robust systematic generalisation of unseen formal instructions in out-of-distribution (OOD) scenarios.

Last, we introduce the first DRL framework that finds diverse competitive policies in complex settings such as realistic car-racing simulators. Such framework is capable of effective driving while fulfilling given constraints, e.g. driving at high speeds while not being aggressive when overtaking. Moreover, the presented solution yields robust agents capable of generalising to OOD requirements and physics.

Para Raquel, mi compañera de aventuras a traves de los mares.

Acknowledgements

First and foremost, I want to thank my PhD advisors, Francesco Belardinelli and Murray Shanahan. I can't stress enough how much they supported me through all this journey. First, all of this happened thanks to Francesco, to whom I will be eternally grateful for giving me the life-changing opportunity to pursue a PhD in AI at Imperial College London. But of course, that was only the start. Through all these years Francesco has enabled me to do my best, not only conducting research together but co-supervising amazing students at Imperial and helping me to discover my passion for mentoring. Nevertheless, this journey would have been very different, and for the worse, without the great advice that Murray provided all the way through. While officially appointed as "second supervisor" Murray has acted since day one as a co-supervisor, giving constant feedback and support. I am also very grateful that even today, in my adventures beyond the PhD, I am sharing this ride with him.

I want to also express my deepest gratitude to Alessandra Russo and Tim Rocktäschel for agreeing to be my thesis committee. It was a privilege to receive their insightful feedback on my work.

This PhD thesis took quite a number of villages full of amazing people to be written. For the shake of brevity I want to provide a bullet list with all the people that made this thesis possible in not particular order other than somewhat chronologically shorting in my head:

- My parents Manuel and Maria-Jose, who have always unconditionally believed in me and supported me through this journey. I could never be the person who did this PhD without them.
- My relatives, for being one of the biggest supports of my life. My grand father, Julian who taught me about magic, and my grandmothers Carmina and Pepi. Raul and Cris who helped me finding those so much needed "Mediterraneamente" breaks between papers and supported me while writing some of those summer rebuttals. To all my relatives, whether named here or not, their love and support have carried me through this journey, and I am deeply grateful to each and every one of them.
- Irene, Laura, Manuel and Mario for being my family far earlier than we officially became one.
- Friends from the past that keep me myself, in the order I met them:
 - Pablo and Teresa, for being there since way before I even realized where I wanted to go in life.

- Jose, for all of those workshop marathons at UPM and helping me with C.
 - Luis, for helping me since day one when I started to dive in AI at UPM and advising me to pursue the PhD abroad, I wouldn't be here without his great advice.
 - Enrico, for being an amazing mate for life. Without him letting me run my-first ever RL experiments on my computer while sharing a room in Milan, I wouldn't have make it to Imperial.
 - The Milano gang, in the order I met them: Salvatore, Shreya, Lydia and Hendrick, for broadening my world and my view of life so much, they have greatly contributed to the person I am today.
- My family at Imperial, I was the first PhD of Francesco's new lab, and with Murray mostly at DeepMind, this would have been a very lonely and different experience without them:
 - Javi, for being my first friend in London and showing me tricks of this country
 - Luca and Manon, who I first met at the RL reading group, they soon became my "lab" and best mates at Imperial.
 - Alex, who co-founded with me the RL seminars at Imperial (ICARL), which proved to be one of the best decisions of this PhD.
 - Nicolas, Dominika, Sacha, and Viet, the rest of my french-speaking family at Imperial.
 - Kate, who kept me in touch with the cibersecurity applications of RL.
 - The rest of the Imperial gang:
 - Fabio, for organizing the RL seminars and his invaluable advice on my first steps of the PhD
 - Antoine, for his constant support with ICARL and, together with Manon, Luca and Maxence, for bringing quality and diversity to my life.
 - The remaining co-organizers of ICARL, Hannah and Maksim.
 - Amani, for being the first point of support through the entire PhD, always responsive and the kindest, I honestly cannot stress well enough what a key role she has in the Department of Computing.
 - All the people I had the privilege to discuss about research through the RL and ICARL reading groups, rep chats or other various circumstances at Imperial, including Bryan,

Callum, Daniel, Edward, Filippo, Georgios, Harry, Kai, Kamil, Mass, Maxime, Myles, Nihir, Norman, Paul, Pedro, Pierre, Rodrigo, Shibe and Stephen.

- All the great researchers I that came to the ICARL events I organized and spent great days with, I learned a lot from all of them broadening my knowledge and expertise in AI: Jakob Foerster, Angeliki Lazaridou, Haitham Ammar, Roberta Raileanu, David Abel, Sam Devlin, Felix Hill, Tim Rocktäschel, Alhussein Fawzi, Elise van der Pol, Alessandro Lazaric, Jessica Hamrick, Lucy Checke, Florian Fuchs, Andrew Lampinen, Daniel Mankowitz, Andrea Michi, Razvan Pascanu, Ilija Bogunovic, Nathan Grinsztajn, Tristan Kalloniatis, Katja Hofmann, Leonard Bauersfeld, Marta Garnelo, Eugene Vinitsky, Jake Bruce, Sebastian Risi.
- During my research internships I also had the opportunity to meet countless talented researchers and engineers. First, from Meta, I am grateful to Oleksandr SavsunenkoView, Alexandra Darnon, Ares MerouehView, Yiwei ChenView, Bharat Thakrar, Alexander KostitsynView, Daniel KohlsdorfView, Gokhan Uzunbas.
- My later research experience at Sony AI had an even deeper impact in this PhD. Without the people listed below there wouldn't have been a chapter about racing cars in this work.
 - First and foremost, I am grateful to Francesco Riccio, my partner in crime for bringing quality-diversity to racing. He was my biggest support in that chapter.
 - Florian Fuchs, my personal GT expert
 - Alisa Devlic, who offered constant support and a continuous stream of research discussion and paper findings.
 - The GT team: Patrick MacAlpine, Takuma Seno, Peter Stone, Kaushik Subramanian, Pete Wurman, Kenta Kawamoto
 - The rest of my colleagues at Sony research: Leon Barret, Roberto Capobianco, Rory Douglas, Alberto Gianmarino, Daniel Hernandez-Perez, Piyush Khandelwal, Raksha Kumaraswamy, Andreeanne Lemay, James MacGlashan, Shruti Mishra, Dustin Morrill, Declan Oller, Radosav Sekulic, Jaden Travnik, Thomas Walsh, Varun Kompella, Yunshu Du.

Finally, I want to express my deepest gratitude to the person who deserves the most thanks for their invaluable support in this work, Raquel, to whom this PhD is dedicated to. Her love and kindness over the past 13 years have been invaluable in helping me find my true self and

providing the inspiration to pursue and write this thesis. More than anyone, she has endured the sacrifices of supporting me pursuing my dreams. This PhD has been a challenging yet the most wonderful chapter of our lives. Te quiero mi loquilla, gracias por hacer de esta vida algo único. Este viaje no ha hecho más que comenzar y sé que lo que nos espera será algo mágico porque estaremos juntos.

Declaration of Originality

This statement is to certify that the work presented in this thesis is my own original research. Collaborators involved in the different projects have been acknowledged. Finally, this thesis is not being submitted for any other degree.

Borja Gonzalez Leon

October 14, 2024

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). 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 and do not use it, or any derivative works, for a commercial purpose. 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.

What drove me into Artificial Intelligence? For me, it's about fulfilling a promise I made to my young self – to unleash real magic into this world. There is nothing as effective as intelligence and learning to unlock the impossible.

CTDE	Centralized Training for Decentralized Execution
CNN	Convolutional Neural Network
CMDP	Contextual Markov Decision Process
dec-POMDP	Decentralized POMDP
DFA	Deterministic Finite Automata
DL	Deep Learning
DQL	Deep Q-Learning
DP	Dynamic Programming
DRL	Deep Reinforcement Learning
EMG	Extended Markov Games
LSTM	Long-short term memory
<i>LTL</i>	Linear-Time Temporal Logic
MARL	Multiagent Reinforcement Learning
MDP	Markov Decision Process
MG	Markov Game
MLP	Multilayer Perceptron
NN	Neural Network
OOD	Out of Distribution
POMDP	Partially Observable MDP
ResNet	Residual Neural Network
RL	Reinforcement Learning
RL-TF	RL with Temporal Logic
RNN	Recurrent Neural Network

RSSM	Recurrent State Space Model
S4	Structured State Space sequence model
SARL	Safety-Aware Reinforcement Learning
SSM	State Space Models
SFs	Successor Features
TD	Temporal Difference

Contents

Abstract	i
Acknowledgements	v
Declaration of Originality	ix
Copyright Declaration	x
1 Introduction	1
1.1 The Journey of Artificial Intelligence	1
1.2 Towards Systematic Learners	4
1.3 Thesis Organisation	9
1.4 Publications	11
2 Technical Preliminaries	13
2.1 Reinforcement Learning	14
2.1.1 Mathematical Foundations	14
2.1.2 Approximating the Value Functions	18
2.1.3 Policy Gradient	18

2.1.4	Model-Based RL	20
2.1.5	Multi-agent Reinforcement Learning	21
2.2	Deep Learning	22
2.2.1	Artificial Neural Networks	23
2.2.2	Deep Layers	25
2.2.3	Activation and Probabilistic functions	31
2.3	Deep Reinforcement learning	33
2.3.1	Q-learning and DQL Variants	33
2.3.2	Advantage Actor Critic	35
2.4	Safety-Aware Reinforcement Learning	36
2.4.1	Constrained Reinforcement Learning	37
2.4.2	Formal Methods	37
2.4.3	Shielded Reinforcement Learning	39
3	Related Literature	41
3.1	Safety-Aware Reinforcement Learning	41
3.1.1	Constrained Reinforcement Learning	42
3.1.2	Shielded Reinforcement Learning	43
3.2	Systematic Generalisation in Deep Reinforcement Learning	45
3.3	Neuro-Symbolic Reinforcement Learning	47
3.4	Policy Exploration	48
3.4.1	Policy Diversity	48
3.4.2	Universal Estimators	50

I	Extending Non-Markovian Formalism to Complex Settings	51
4	Solving Complex Non-Markovian tasks through MARL	52
4.1	Preliminaries	53
4.2	Non-Markovian Multi-Agent Specifications	56
4.2.1	Non-Markovian Reward Games	56
4.2.2	Extended Markov Games	58
4.3	Deep MARL with <i>co-safe LTL</i> Goals	62
4.3.1	I-DQL with <i>co-safe LTL</i> Goals	62
4.3.2	I-LPOPL	63
4.4	Experimental Evaluation	65
4.4.1	Experimental Setup	65
4.4.2	Specification Sets	66
4.4.3	Experimental Results	67
4.5	Discussion	69
5	Non-Markovian Task Execution Under Imperfect Information	70
5.1	Partially Observable Models	71
5.1.1	Navigating PONMRDPs	73
5.2	Extended POMDPs	75
5.3	A Counterexample to a Naive Extension	79
5.4	Discussion	82

II	Drivers of Systematic Generalisation in Deep Reinforcement Learning	84
6	Emergence of Grounded Systematic Generalisation with Abstract Operators	85
6.1	Solving Zero-Shot Instructions in Task Temporal Logic	87
6.1.1	Task Temporal Logic	88
6.1.2	The Symbolic Module	90
6.1.3	The Neural Module	93
6.2	Experiments	96
6.2.1	Experiment Settings.	96
6.2.2	Empirical Results	99
6.2.3	The DRL struggle with negated instructions	104
6.3	Discussion	106
7	Generalising through Latent-Goal Architectures	108
7.1	Learning to solve SATTL instructions	110
7.1.1	Safety-aware Task Temporal Logic	110
7.1.2	Neuro-symbolic agents	112
7.2	Latent-goal architectures	113
7.3	Experiments	116
7.3.1	Empirical setting	117
7.3.2	Experiment Results	122
7.3.3	Ablations	125
7.3.4	Systematic Learning	127
7.4	Discussion	129

III	Safety-aware Policy Exploration	131
8	Discovering Creative Safety-Aware Policies with DUPLEX	132
8.1	Notation	135
8.2	Learning Near-Optimal Diverse Behaviors in Multi-Goal Continuous Settings	136
8.2.1	Context-conditioned Diversity Learning	137
8.2.2	Estimating Successor Features	140
8.2.3	Training DUPLEX	142
8.3	Experiments	143
8.4	Discussion	148
9	Conclusions and Future Work	149
9.1	Formal Languages and Systematic Generalisation with LLMs	153
	Bibliography	154
A	Appendix for Systematic Generalisation with Abstract Operators	184
A.1	Hyperparameters	184
A.2	Hardware	185
A.3	Translating TTL into LTL_f	185
B	Appendix for Network Architectures with Latent Goals	187
B.1	Symbolic Module	187
B.2	Additional details of empirical results	188
B.2.1	Statistical Analysis	189
B.3	SATTL and LTL_f	191

C	Appendix for Policy Exploration	194
C.1	Environment details	194
C.2	Ablation in USFA	195
C.3	Hyperparameters	195
D	Appendix for Extended Markov Games	197

List of Tables

4.1	Computing times of the different algorithms and variants studied in this chapter.	69
6.1	Results from 500 BCMs in Minecraft (MC) and MiniGrid (MG) with negated instructions and the most challenging disjunction-based tasks. In “Reliable” higher is better while in “Deceptive” lower is better. Rewards are normalised so that 1.0 is the performance of a random agent. Agents show a stronger systematic generalisation the bigger the difference between respective reliable and deceptive instructions. Pos. and neg. labels refer to atomic positive or negative instruction respectively, pointing to one of the two objects present. In disj. 2 nd the instructions refers to two objects, but only the second is present. Disj. 2 points to two objects present in a map with 4 items. Results are bolded where both rewards with reliable instructions is greater than 2.0, i.e., twice the rewards of a random agent, and less than 1.0 with deceptive instructions, i.e., the agent succesfully exhibits a policy that avoids the rewarding object when the intruction says so.	101
6.2	Complete results in BCMs with disjunctions. Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.	102
6.3	Results from 200 maps with test complex instructions, i.e, longer than training instructions composed only by zero-shot tasks. Steps refers to the number of actions per instruction needed by the agents. The highest value over all networks is bolded.	103
6.4	Complex instructions used in Table 6.3. The 6 objects included in these instructions belong to the test set (X_3) in the Minecraft-inspired environment, that means every intrusion here is zero-shot.	103

6.5	Results from a ResNet in the Minecraft setting when using a validation set or when not, i.e., the latter are the results from an agent where we did not perform early stopping and overfitted. We see that the different behaviours with positive and negative when the agent is overfitting align with the results from Hill et al. (2020). Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.	105
7.1	Study on compositional learning with zero-shot objects and instructions. Results show the performance obtained by each network in 200 test maps when rewards are given according to the “real goal”, while the symbolic module provides the “given instruction” to the neural module. An agent learning compositionally should have 1^{st} row $>$ 2^{nd} row $>$ 3^{rd} row for the values within its column. Also, values lower than random are only acceptable in the third row (deceptive instruction). The mean performance of a random walker is 12.5 independently of the “given instruction”.	127
B.1	Results with either training (train) or OOD instructions (test) in 500 maps (per size) of different dimensions. Note that sizes 14 and 22 are OOD for all the agents. MC refers Minecraft whereas MG to MiniGrid. Results show the average reward and standard deviation from 10 independent runs (i.r.). Values are normalised so that 100 refers to the highest performance achieved by the best run globally in maps of the given size and benchmark. Best average rewards in each setting and size is bolded.	191
B.2	Results from 5 i.r. of a vanilla BRIM (BRIM), a BRIM with residual connection (ResBRIM) and BRIM with latent goals (BRIM ^{LG}). Best results for each setting and size are in bold. . . .	191
B.3	Ablation studies in Minecraft (5 i.r. per variant). PrediNet ^{LG} _{noSub} and PrediNet ^{LG} _{noPos} are variants of PrediNet ^{LG} without the element-wise subtraction and the feature coordinates respectively. PNMHA ^{LG} uses an MHA in CM2 while MHAPN ^{LG} uses an MHA in CM1. Best results are in bold	192
B.4	Ablation studies with PrediNet ^{LG} in Minecraft (5 i.r. per variant). In the first set we explore the impact of different bottleneck sizes. In the second we contrast having L_s task-agnostic or not. For every ablation set, best results are bolded.	192
B.5	Two-way ANOVA test with Alpha= 0.05 in Minecraft. Neural network refers to the impact of the different neural networks in the central modules, whereas architecture configuration refers to the impact of whether using a latent goal or a standard configuration.	193

B.6	Two-way ANOVA test with $\text{Alpha} = 0.05$ in MiniGrid.	193
C.1	Hyperparameters of DUPLEX.	196

List of Figures

1.1	Interaction with GPT 4. The model is given a variant of the Tower of Hanoi to solve. GPT 4 correctly identifies the nature of the problem and starts well with the first actions. However, it fails in move 10 forgetting about the hair tie 2 and later starts making erratic moves.	3
2.1	The RL framework from Sutton et al. (1998).	14
2.2	Illustration of the simplest type of NN, an MLP. This one has an input layer with four artificial neurons followed by two hidden layers of six neurons each that terminates in an output layer of two neurons.	23
2.3	Depiction of a residual connection. The information from the residual path is typically incorporated to the standard flow of information through addition or concatenation.	26
2.4	Illustration of an Autoencoder N layers for encoder and decoder, respectively.	27
2.5	Example of a convolution with a 2x2 kernel with one channel. Note that stride is two since we move the kernel two positions for every operation. This example uses a padding of dimension 1 reflected by the zeros surrounding the input.	28
2.6	Illustration of how an RNN iterates through time.	29
2.7	Illustration of a self-attention layer, i.e., an attention layer that obtains the relationship between the different pieces within the input.	30
2.8	The Shielded RL framework	40
3.1	Types of shielding as first presented by Alshiekh et al. (2018). Left is an example of preemptive shielding. Right, is post-posed shielding.	43

4.1	Rewards obtained by DQN-I and I-DQN-I greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 50k training steps per specification of the set.	66
4.2	Rewards obtained by LPOPL and I-LPOPL greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 25k training steps per specification of the set.	68
5.1	The models for the PONMRDP (left side) and an extended POMDP (right side) following Def. 5.7. Red lines denote some $\tau(es)$ transformations while blue lines we illustrate the cases in which $\sigma(s)$ is only the current state.	76
5.2	Counterexample to naive extension. The models for the PONMRDP (left side) and its extended POMDP (right side), satisfying item 5' instead of the proposed item 5 from Definition 7. Red lines denote some $\tau(es)$ transformations while blue lines illustrate the cases in which $\sigma(s)$ is only the current state.	81
6.1	Common features of the neural network architectures evaluated in Section 6.2. All the variants tested in the Minecraft-inspired environment follow the architecture depicted on the left. The one on the right is the structure employed in Minigrid. $\pi(a_t)$ and V_t refer to the output actor and critic layers respectively Mnih et al. (2016).	93
6.2	The different convolution architectures explored in this chapter. The different number of channels in the last layer of the Env-CNN architectures is done to provide evidence that such feature does not affect how the agent generalises.	94
6.3	Left. Procedurally generated map in the Minecraft-inspired environment with its corresponding extended observation (right) when the agent's task is to reach either iron or an axe (top row). Tasks are specified in the top row of the extended observation in this setting by depicting the object itself.	96
6.4	Procedurally generated map in MiniGrid where the task is to reach an object different from a grey ball, i.e., " <i>not</i> a grey ball". Here instructions are given on a separated channel and objects are combinations of shapes and colors.	97
6.5	Train and validation plots with the different population sizes and 5 independent runs. Light colors refer to performance with training objects, whereas dark refer to validation objects. Lines refer to the median while shaded areas represent the 25-75 quartile difference.	100

- 7.1 Schematic representation of **a)** typical architecture present in previous literature, **b)** *latent-goal* architecture we propose in this chapter. Key design features of our architecture are that the latent-goal (L_g) is the output of an information bottleneck while the latent representation of the observation L_o is task-agnostic. 113
- 7.2 Neural network architectures used in the main experiment sections. **Top:** standard architecture from previous literature. The central module can be either fully-connected or relational layers. Outputs $\pi(a_t)$ and V_t refer to the actor and critic respectively (Mnih et al. 2016). **Bottom:** proposed latent-goal architecture. 115
- 7.3 **Top:** vanilla BRIM and ResBRIM (without and with the residual connection respectively). Green cells within the BRIM layer refer to activated modules, i.e., those selected by the top-k attention, while gray cells are deactivated. The bottom layer receives as input the output of the central module and the hidden state from the upper layer in the previous time step (h'_{t-1}). The upper layer receives the output from the bottom layer (h_t). Only strong connections (the ones between activated modules) are shown. In the case of ResBRIM the upper layer also receives the output of the Central module. **Bottom:** BRIM^{LG}. The architecture is similar to the ResBRIM but the top layer receives as additional input an embedding from the observation exclusively (L_o). Consequently, the top layer has only access to the current goal through the lower layers of the hierarchy. 116
- 7.4 **Top:** a 7x7 Minecraft map (left) and its corresponding extended observation (right) with the visual instruction “move through either soil or stone until reaching an axe or a sword”. Tasks are specified at the top of the extended observation by depicting objects and SATTL operators. Each tile has a resolution of 9x9 pixels. **Bottom:** a 22x22 map in MiniGrid where the agent needs to “avoid orange lava until reaching a gray or dark-green key”. Instructions are given through a text channel and each tile has 8x8x3 pixels. Highlighted cells indicate the field of vision of the agent. 117
- 7.5 Training plots of the networks studied in Section 7.3.2. Continuous lines correspond to the 50th percentiles while the shadowed areas are the 25th and 75th percentiles. Note that good train performances do not yield to similarly good test results as detailed in the main text. **Left:** Minecraft training plots. **Right:** MiniGrid plots. Note that the general performance drop at 20M in this setting is motivated by the higher chance of generating larger training maps from that point on. 122

7.6	Results with training (top) and out-of-distribution (OOD) instructions in 500 maps (per size) of different dimensions. Sizes 14 and 22 are OOD. MC refers Minecraft whereas MG to MiniGrid. Results show the average reward and standard deviation from 10 independent runs. Values are normalised so that 100 refers to the highest performance achieved by the best run globally in maps of the given size and benchmark.	123
7.7	Results of 5 independent runs of vanilla BRIM (BRIM), BRIM with residual connections (Res-BRIM), and with LGs (BRIM ^{LG}).	124
7.8	The PrediNet from Shanahan et al. (2020).	124
7.9	The MHA from Zambaldi et al. (2018).	125
7.10	Ablation studies in Minecraft (5 i.r. per variant). PrediNet ^{LG} _{noSub} and PrediNet ^{LG} _{noPos} are variants of PrediNet ^{LG} without the element-wise subtraction and the feature coordinates respectively. PNMHA ^{LG} uses an MHA in CM2 while MHAPN ^{LG} uses an MHA in CM1.	126
7.11	Ablation studies with PrediNet ^{LG} in Minecraft (5 i.r. per variant). Left: Impact of different bottleneck sizes. Right: Impact of having L_s task-agnostic or not.	127
8.1	DUPLEX data flow training a multi-policy agent over a set of tasks. At each iteration of DUPLEX we provide three inputs to give context about the current operational dynamics and task requirements c ; an encoding of the current policy being used z and the current state of the environment s . Our agent then is trained to output three quantities: critic estimates for the intrinsic and extrinsic rewards and successor features to drive diverse behavior discovery. Finally, the algorithm samples policies in Π uniformly and rolls them out to collect more experience. . . .	133
8.2	Results in Mujoco Walker (Top) and Ant (bottom) multitask environments. On the y-axis we report the reward accumulated by the agent while on the x-axis diversity according to Def. 8.1. Ideally, we want to be as close as possible to the top right region. OOD dynamics refers to gravities that are at least 40% stronger or weaker than the strongest and weakest gravities seen in training, respectively. OOD tasks refers to tasks were the agent needs to walk forward below a height 20% lower than the lowest height seen in training tasks. Dots and lines represent mean and standard deviation, respectively. Note that only DUPLEX and UVFA are evaluated in Test since the other algorithms already fail at training.	143

8.3	Results in GT. Top is the reward-diversity trade off, DUPLEX/DOMiNO_best_policy $\max_{\pi \in \Pi}(\pi \in \Pi)$ of every run, while DUPLEX/DOMiNO_avg refers to $\text{avg}_{\pi \in \Pi}(\pi \in \Pi)$ of that run. We see that only DUPLEX is able to generate <i>both</i> competitive and diverse behaviors. The remaining figures illustrate the number of active policies (middle), i.e., number of policies that are finishing laps, and min lap times of the diverse policies (bottom). Note that the number of active policy is important to showcase that all policies are actively searching for diverse behaviors through different epochs. In fact, our target is having all policies active with diverse lap times (diversity) below the 150 seconds mark (near-optimality). We find that applying our soft lower bound ($\beta \neq 0$) is key to focus diversity in the region of interest.	145
8.4	In a Walker single-task settings, both DUPLEX and DOMiNO do not estimate SFs and use $d_{\pi_c}^{\text{avg}}$ instead. On the left, we show results from the best policy in each variant while on the right, the average reward across all policies.	146
8.5	Ablation of DUPLEX in Multitask Walker when 1) not using entropy (H) in the ψ^δ objective from Eq. 8.7; 2) using $\min(\psi_{\theta_{1,2}}^\delta)$ - as in SAC - instead of $\text{avg}(\psi_{\theta_{1,2}}^\delta)$; 3) eliminating the upper bound trigger in α_χ ; 4) using ψ^{avg} instead of ψ^δ	147
C.1	Ablation on USFA, we see that incorporating entropy to the learning objective of the successor features estimator as in DUPLEX yields a significant boost to accumulate rewards.	195

Chapter 1

Introduction

1.1 The Journey of Artificial Intelligence

Building human-like intelligent machines is a major challenge for science and engineering. The idea of replicating human intelligence has been an evolving topic that has captivated societies for millennia. Already in ancient Greece, with Hephaestus—the blacksmith god—and his creation (Gao 2020), or in Jewish folklore (Gelbin 2021), we find examples of AI in the form of the Golem, a fabricated creature typically made of mud or clay, capable of answering any question posed to it. This recurrent fascination with creating life-like objects reflects the human desire to mimic intelligence and life in machines. This desire materialised in the 18th century with the construction of sophisticated automata, such as the digestive duck and the chess-playing mechanical Turk (Riskin 2003). The transition from mere craftsmanship to the realm of programmable computers marked another significant leap in this quest. Pioneers like Charles Babbage and Ada Lovelace envisioned machines that could transcend basic calculations, laying the groundwork for modern computing and the concept of mechanised intelligence. Alan Turing’s groundbreaking work (Turing 2009), including the Turing machine and the Turing Test, further pushed the boundaries, exploring the possibility of machines exhibiting intelligence indistinguishable from that of humans.

The term “artificial intelligence” was coined in the mid-20th century, marking the inception of a field

dedicated to developing computer systems capable of performing tasks traditionally associated with intelligent beings. The Dartmouth workshop of 1956 is celebrated as the birthplace of AI, spearheaded by John McCarthy and other pioneers. Early AI research emphasised symbolic computation and reasoning, with activities such as chess-playing serving as benchmarks for intelligent behavior. The 1997 victory of IBM's Deep Blue over chess champion Garry Kasparov represented a significant milestone, yet it also underscored the limitations of AI at the century's end. At that time, it was speculated that AI would require a century to surpass the best human players at Go, a game known for its complexity and the depth of human intuition needed for mastery. Contrary to these predictions, the AI landscape underwent a dramatic transformation with the advent of innovative learning-based techniques. In 2016, Google DeepMind's AlphaGo made headlines by defeating Lee Sedol, one of the world's leading Go players. AlphaGo's (Silver et al. 2016) triumph was revolutionary, demonstrating that AI algorithms could achieve victory through strategies once deemed flawed by humans. This opened avenues for the exploration of new knowledge and strategies in Go, a game humanity had spent approximately 4,000 years attempting to master.

Since then, the field of AI has made significant strides beyond the realm of games. A notable achievement was realised in 2020 with the development of AlphaFold (Jumper et al. 2021), an AI system engineered to address the protein folding problem—a monumental challenge in biology. AlphaFold's capability to accurately and swiftly predict the 3D structures of proteins had profound implications in biochemistry and medicine, paving the way for advancements in understanding diseases and facilitating drug discovery. More recently, a groundbreaking development emerged in late 2022 with the introduction of ChatGPT, a web-based application powered by a Large Language Model (LLM) (Brown et al. 2020). These models, trained on massive datasets, are adept at generating coherent and contextually appropriate text, answering questions, summarising information, and even crafting creative content, thereby resembling the Golem of ancient tales. However, unlike their mythical counterpart, LLMs are non-physical entities, and they struggle with basic tasks when presented with slight variations of grounded scenarios encountered during training. For example, when querying the most sophisticated LLM with the Tower of Hanoi problem (Kotovsky et al. 1985), they can solve the task, creating an illusion of mastering the underlying cognitive skills. Yet, constructing a variant of the problem as shown in Figure 1.1 reveals that, unlike humans, they have not systematically learned to

solve this problem, highlighting a fundamental limitation in current AI capabilities.

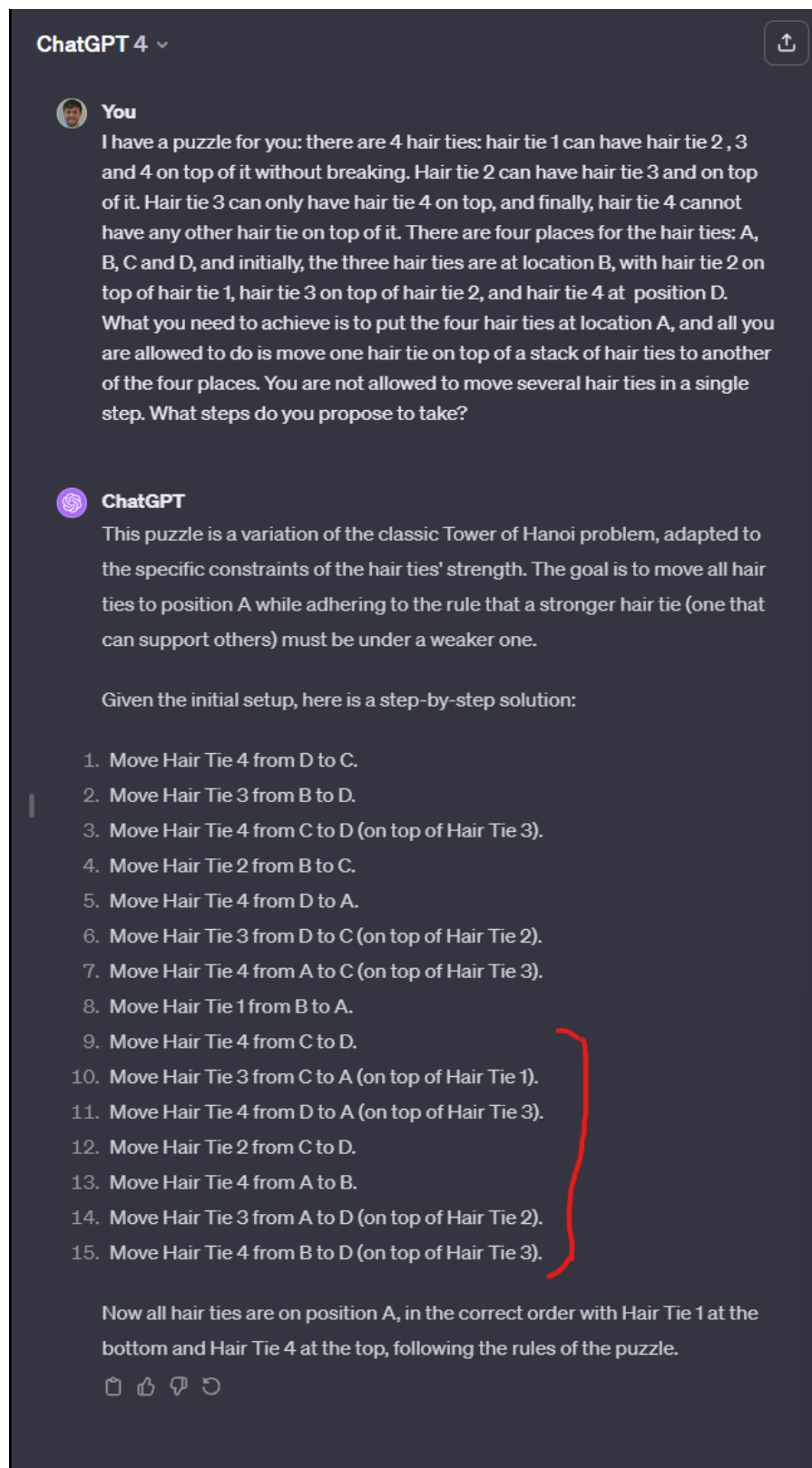


Figure 1.1: Interaction with GPT 4. The model is given a variant of the Tower of Hanoi to solve. GPT 4 correctly identifies the nature of the problem and starts well with the first actions. However, it fails in move 10 forgetting about the hair tie 2 and later starts making erratic moves.

1.2 Towards Systematic Learners

Machine Learning. Traditional coding methods often fall short in generality and face difficulties with tasks involving many interacting elements. The breakthrough came with the realisation that developing a general program capable of improving its performance over time could be more practical than attempting to solve a specific task directly. This process of automatic enhancement is akin to biological learning, leading to the term *machine learning* being coined for these algorithms (Bishop 2006). The simplest form of machine learning is *supervised learning* (Hastie et al. 2009), where algorithms learn from a dataset of input-output pairs. The objective is to train these algorithms to map inputs to their correct outputs. This mapping, known as a model, is what gets trained, and a loss function evaluates the accuracy of the model's predictions against the actual outputs. In supervised learning, the model and the loss function are typically differentiable, allowing for the adjustment of model parameters through *gradient descent* (Kiefer & Wolfowitz 1952) by calculating the gradient of the loss with respect to the model parameters and updating the parameters in the direction that reduces the loss. Another category is *unsupervised learning* (Ghahramani 2003), also referred to as *self-supervised learning*, utilised when the target outputs are unknown but there's a need to uncover some structure within the data. These algorithms strive to represent the data meaningfully, exemplified by the initial training phase of an LLM, where the model learns to predict the next word in a sequence from vast internet data.

Both supervised and unsupervised learning typically rely on static experimental setups with large, albeit limited, datasets. However, the most natural form of learning observed in nature differs significantly. Animals learn by interacting with their surroundings, forming predictions, and taking actions to achieve goals such as avoiding danger or seeking food, without explicit supervision or a direct loss function to gauge the impact of their actions. To replicate this natural learning paradigm, a third category of machine learning, *reinforcement learning*, was developed. This approach focuses on learning through interaction, where algorithms make decisions based on feedback from their environment to achieve specific objectives.

Reinforcement Learning. In the realm of sequential decision-making, reinforcement learning (RL) presents a unique approach (Sutton & Barto 2018). Unlike methods that rely on predefined target actions, RL employs a reward signal to steer its objectives. This signal doesn't immediately assess the effectiveness of an action at a specific moment but might result from a sequence of actions or decisions made previously, posing a challenge in appropriately assigning credit. Therefore, RL aims not just to maximise immediate rewards but to accumulate rewards over time, focusing on long-term returns. Actions yielding modest immediate rewards are favored if they contribute to greater future rewards. The RL framework typically involves an agent learning to interact with its environment, with inputs as environmental observations and outputs as actions taken in response. The environment is often modeled as a Markov decision process, where transitions depend solely on the current state and the action taken, which emphasises the need for current state information to encapsulate all relevant past details. From the agent's perspective, the environment acts like a dataset to be explored for valuable information, with the gathered data synthesised into functions representing expected values, policies, and models of the environment. While initial RL research utilised simple tabular methods for value mapping, the integration of deep learning has enabled the representation of more complex features in challenging environments.

Deep Learning. Deep learning traces its origins to the perceptron, a parametric machine learning model conceived by Rosenblatt (1958). The Mark I perceptron, designed for image recognition, consisted of 400 photocells randomly connected to *neurons*, forming a basic binary classifier. Initially, the perceptron was met with enthusiasm, as a New York Times article stated it was "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself, and be conscious of its existence". However, the limitations of the perceptron in handling only linearly separable models were underscored in the book "Perceptrons" by Marvin & Seymour (1969). This period also saw early overexcitement in the 1950s with the Georgetown experiment in 1954, which demonstrated the first-ever successful fully automatic translation of more than sixty Russian sentences into English. This led many to prematurely anticipate the resolution of machine translation by the 1960s, though real progress only began in the 2000s. The history of AI is marked by cycles of inflated expectations followed by disappointment. These cycles, known as "AI winters," led

to significant reductions in funding and a dwindling interest from the research community, particularly impacting artificial neural networks. Despite these challenges, research persisted. It was found that configuring two or more perceptrons with intervening non-linearity could represent complex functions. These deeper models could be trained effectively using gradient descent and *backpropagation* (Linnainmaa 1970). A notable advancement was Yann LeCun’s development of convolutional neural networks (CNNs), inspired by the visual cortex. LeCun’s model, trained end-to-end through backpropagation, found practical application in banks for reading handwritten numbers (LeCun et al. 1989). However, a general lack of computational resources stifled interest in artificial neural networks until 2012, when AlexNet (Krizhevsky et al. 2012), a deep convolutional network, significantly outperformed other models in the ImageNet Large Scale Visual Recognition Challenge. This reignited interest in combining large datasets, deep neural networks, end-to-end backpropagation training, and hardware acceleration, like GPUs. Their combination reached another critical point in 2017 with the introduction of “Attention Is All You Need” (Vaswani et al. 2017), which introduced the *transformer* architecture. Using a mechanism called self-attention, transformers significantly improved performance in various tasks, especially in natural language processing, becoming foundational in modern deep learning and advancing language models like BERT (Devlin et al. 2018) and GPT (Radford et al. 2019). However, their application to continual learning problems, such as those addressed in reinforcement learning, while promising, has not yet achieved the same level of success.

Deep Reinforcement Learning. Reinforcement learning has advanced significantly with the integration of artificial neural networks as sophisticated parametric function approximators. A notable early instance of this integration was TD-Gammon (Tesauro 1994), which merged reinforcement learning with a deep neural network to become proficient in the game of Backgammon through self-play. The true emergence of deep reinforcement learning (DRL), however, is often credited to the groundbreaking work “Playing Atari with Deep Reinforcement Learning” (Mnih et al. 2013). This study introduced Deep Q-learning (DQL), an innovative algorithm that learned control strategies directly from complex sensory inputs (pixels) using reinforcement learning. The paper, along with its improved 2015 version (Mnih et al. 2015), established various principles for agent and environment design, as well as infrastructure, that have become fundamental in the field. DQL reached human-level

or superior performance in several Atari 2600 games from pixel input, paving the way for subsequent significant achievements in the field. These include mastering a wide range of games (Silver et al. 2016, Vinyals et al. 2019, Wurman et al. 2022), advancements in physics (Degraeve et al. 2022), and discoveries in mathematics (Fawzi et al. 2022).

Safety-aware (Deep) Reinforcement Learning. Deep reinforcement learning has fallen behind to other machine learning fields in reaching diverse real-world applications. A major hurdle when deploying DRL into the real world is the presence of additional constraints within the control problem. For example, in autonomous driving, while we do not desire a car that drives too slowly, neither would we prefer a car that gets us to our destination a few seconds earlier at the expense of a much higher likelihood of an accident. These limitations have spurred interest in designing new forms of reinforcement learning algorithms that prioritise reliability and safety (García & Fernández 2015a). We refer to safety-aware (deep) reinforcement learning (SARL) as those techniques that aim to optimise a primary objective while adhering to extra constraints imposed on the agents. Examples of such approaches include constrained reinforcement learning, a method that gained traction with the introduction of Constrained Policy Optimisation Achiam et al. (2017). That approach integrates costs and limits into the RL model to address a multi-objective optimisation problem. However, it is limited to operate in settings where a significant number of constraint violations during training is permissible. Concurrently, an alternative strand of research was initiated by Alshiekh et al. (2018), proposing the integration of a proxy or *shield* between the agent and the environment. This shield permits the learning agent to perform actions only when they are considered safe by the shield. Nonetheless, shielding approaches often depend on stringent assumptions regarding the user’s prior knowledge about the environment, severely limiting their practicality. Despite these efforts, both techniques generally suffer from a lack in terms of generalisation, robustness and exploration of the state space. Thus, in this work we make contributions in fundamental components of DRL, with special focus on those limitations.

Temporal Logic and Reinforcement Learning. Applying RL to autonomous agents executing formal specifications has become a thriving area of research in SARL (Brafman et al. 2018, Jothimu-

rugan et al. 2019, Camacho et al. 2019, Xu et al. 2020, Cohen et al. 2023). Thus, our research on grounded language learning also focuses on learning instructions given in temporal logic (TL) Huth & Ryan (2004). While earliest works combining RL and TL focused on tackling temporally extended goals, naturally described as a non-Markovian reward decision process (NRMDP), by generating an equivalent extended MDP where RL algorithms can be applied (Bacchus et al. 1996). Growing interest in DRL sparked research to investigate increasingly diverse and expressive languages (Brafman et al. 2018, Toro Icarte et al. 2018). However, those works use formalism based on Bacchus et al. (1996), which is restricted to single-agent, fully-observable environments. Another focal point of this work is introducing new formalism to eliminate those restrictions.

Systematic Generalisation. Systematicity describes a characteristic of human cognition, signified by the ability to understand a concept and inherently grasp ideas with related semantic meanings (Fodor & Pylyshyn 1988). For example, an individual who understands the phrase “speak quietly” will naturally comprehend “walk quietly”, assuming they know how to walk. It implies that if a model can handle specific elements (or words) in one configuration, it should likewise manage the same elements in different configurations. This ability is also referred to as *combinatorial* or *compositional generalisation* in certain research (O’reilly 2001, Battaglia et al. 2018, Lake 2019). However, it’s acknowledged that human reasoning is not always perfectly systematic (O’Reilly et al. 2013). Therefore, the emphasis in artificial models has shifted towards assessing how well a system generalises in accordance with the systematic structure intrinsic to the problem at hand. Recent efforts in neural networks concerning systematicity have focused on uncovering optimal methods to encourage such behaviors in trained models Lake & Baroni (2023). A principal theme of this thesis is the emphasis on systematic generalisation, as it facilitates safety in a manner inspired by human learning—similar to how a child learns to avoid harmless objects while playing in a playground and can apply this knowledge to navigate more hazardous situations.

Diversity. Strong generalisation is closely linked with effective exploration (Jiang et al. 2024). However, reinforcement learning, especially SARL, faces challenges in exploring robust solutions and maintaining control robustness (Odriozola-Olalde et al. 2023, Zhao et al. 2023). Complex en-

vironments often exhibit non-stationary characteristics, necessitating agents capable of executing diverse and flexible behaviors adaptable to various contexts and opponents (Vinyals et al. 2019, Zahavy et al. 2024). To address this, an increasingly large community has begun investigating methods to train RL agents that can solve tasks with a wide range of near-optimal strategies. Unlike standard RL approaches (Haarnoja et al. 2018, Espeholt et al. 2018), diversity learning focuses on training a spectrum of behaviors to address a set distribution of tasks (Zahavy et al. 2023, Kumar et al. 2020, Cully et al. 2015). These frameworks yield agents that not only find varied solutions to a problem but also excel in environment exploration. Nonetheless, existing methods either fall short in generalising to novel tasks (Zahavy et al. 2023, Kumar et al. 2020) or rely on evolution as an optimisation technique, which can be computationally intensive (Cully et al. 2015, Flageat et al. 2023). Another focal point of this thesis is mitigating these challenges within the diversity of RL. We approach this issue by utilising more general forms of function approximators, termed *universal-estimators* (Schaul et al. 2015, Borsa et al. 2019), which we enhance to discover successful strategies for highly realistic car racing simulators.

1.3 Thesis Organisation

The various projects undertaken during this PhD have focused on the themes of systematic generalisation and robustness in autonomous agents through deep reinforcement learning, and on grounded language learning from temporal logic instructions. Our efforts have strengthened the field of RL in general, and safety-aware reinforcement learning in particular, by addressing key challenges inherent in SARL. The organisation of this work does not follow a chronological sequence but is instead categorised by the topics to which they contribute. These areas encompass systematic generalisation, safety-aware policy exploration, and theoretical frameworks for RL with temporal logic to multi-agent and partially-observable settings. The subsequent sections of this thesis are structured accordingly:

Preliminaries. The first two Chapters of the Thesis presents the building blocks to understand and contextualise our work. We start with Chapter 2 introducing the technical background and the main notions to understand this PhD. Then we contextualise our work in Chapter 3 starting with a general

overview of literature about SARL and continue with a more specific review of the different topics that we cover through this thesis: systematic generalisation, policy exploration and neuro-symbolic RL.

Extending non-Markovian formalism to complex settings. Many works combining formal methods and RL are based on a formalism that enables non-Markovian settings for RL algorithms but that is limited to fully-observable and singular-agent settings (Bacchus et al. 1996). Part I is devoted to two predominately theoretical works that introduce novel mathematical formulations of non-Markovian models to be directly applied to RL algorithms in multi-agent (Chapter 4) and partially-observable settings (Chapter 5).

Drivers of systematic generalisation in deep reinforcement learning. Part II presents our work in the field of systematic generalisation in deep reinforcement learning. We start with Chapter 6 introducing a novel neuro-symbolic framework that is capable of systematic generalisation of complex multi-task goals and provide the first evidence that deep reinforcement learning agents can generalise abstract logic operators such as negation as early as with positive instructions, that is, from six examples. Previous published research (Hill et al. 2020) had stated that DRL agents required more than ten times additional diversity in their training set. Chapter 7 introduces a novel deep learning configuration applicable to a wide range of state-of-the-art deep learning architectures and demonstrates how our design improves robustness and out-of-distribution (OOD) generalisation capabilities of any of these architectures for DRL agents.

Safety-aware policy exploration. Part III targets policy diversity and exploration under constrained objectives in complex settings. Chapter 8 we present a novel framework capable of training diverse high-performing multi-goal policies that satisfy user-defined constraints while exhibiting strong OOD generalisation as well. We demonstrate that our algorithm is the first one to exhibit diverse near-optimal policies in the physically realistic, car-racing simulator GranTurismo™ 7.

Conclusions. We finish with Chapter 9 that discusses the main conclusions extracted from this research and the most promising lines to keep advancing the field.

1.4 Publications

The following is a list, in chronological order, of the successful research I completed through my doctoral studies. Works that are directly related to a chapter in this thesis include corresponding references. My contributions are stated for the works where I collaborated with people beyond my PhD supervisors.

1. **Borja G. Leon**, and Francesco Belardinelli. “Extended Markov Games to Learn Multiple Tasks in Multi-Agent Reinforcement Learning.” European Conference of Artificial Intelligence. 2020. Chapter 4 is based on this work.

2. Belardinelli, Francesco*, **Borja G. León***[†], and Vadim Malvone*. “Enabling Markovian Representations under Imperfect Information.” International Conference on Agents and Artificial Intelligence 2022. Chapter 5 is based on this work.

* Means authors are in alphabetical order. [†] means corresponding author. I spearheaded this theoretical-only project formulating the main ideas, theorems and proofs. Also led the writing of the manuscript.

3. **Borja G. León**, Murray Shanahan, and Francesco Belardinelli. “Agent, do you see it now? systematic generalisation in deep reinforcement learning.” ICLR Workshop on Agent Learning in Open-Endedness. 2022. Chapter 6 is based on this work.

4. **Borja G. León**, Murray Shanahan, and Francesco Belardinelli. “In a Nutshell, the Human Asked for This: Latent Goals for Following Temporal Specifications.” International Conference on Learning Representations. 2022. Chapter 7 is based on this work.

5. **Borja G. León**, Francesco Riccio, Kaushik Subramanian, Peter R. Wurman, Peter Stone. “Discovering Creative Behaviours through DUPLEX: Diverse Universal Features for Policy Exploration”. Under Review. 2024. Chapter 8 is based on this work

This is a work that I led through an internship in the Game AI team at Sony AI. I spearheaded this project proposing the main ideas and designing the algorithm that we introduce here. While I used the existing codebase at Sony, I implemented all the functions and code that are particular of the proposed algorithm, the baselines – except for SAC and QRSAC – and the novel environments presented here. I also did all the experiments and led the writing of all the sections of the manuscript. Francesco Riccio proposed the original research problem and acted as my manager while providing useful advice through the internship. He also participated actively in the writing of the manuscript helping me integrating feedback from the research group. The rest of the authors provided iterative feedback on the manuscript and directed – at different levels – the research group that this project belongs to.

Chapter 2

Technical Preliminaries

In this chapter we cover the prerequisites knowledge to follow this work. Particularly, we start with Section 2.1 diving into the topic of reinforcement learning (RL), briefly explaining its foundational mathematical frameworks (Section 2.1), how we estimate the value of states and actions (Section 2.1.2), improve behaviors (Section 2.1.3), learn world models (Section 2.1.4), and multi-agent scenarios (Section 2.1.5). Section 2.2 reviews deep learning. Specifically, Section 2.2.1 introduces the artificial neural networks and their key functioning components, Section 2.2.2 briefly explains the different types of deep layers our work employs and Section 2.2.3 gives an overview of the main activation functions that increase the capacity of deep learning. We continue in Section 2.3 with foundational deep RL algorithms that we use through our work, including Q-learning variants (Section 2.3.1) and Advantage Actor Critic (Section 2.3.2). Last, Section 2.4 explains safety-aware RL starting with a brief introduction about constrained RL Section 2.4.1, then, introducing key concepts from formal methods (Section 2.4.2), and ends reviewing the fundamental concepts of shielded RL (Section 2.4.3).

2.1 Reinforcement Learning

Reinforcement Learning (RL) (Sutton & Barto 2018) is a learning framework where an autonomous agent interacts with its environment to maximise an input signal typically referred to as reward (see Figure 2.1). RL and its extension through deep learning, deep reinforcement learning (DRL) (Mnih et al. 2015), is most commonly used to describe sequential decision making problems that require a mapping from inputs – states – to actions. While this section covers the required insights in RL to follow this work, we refer the readers to Sutton & Barto (2018) for a more general overview.

2.1.1 Mathematical Foundations

Decision-making problems tackled through RL are mathematically grounded as Markov decision processes:

Definition 2.1 (MDP). *A Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle S, A, P, R, \gamma \rangle$ where*

- $s \in S$ is the true state of the environment.
- A is the set of actions $a, a', \dots$
- $P(s' | s, a) : S \times A \times S \rightarrow [0, 1]$ is the (probabilistic) transition function.
- $R(s, a, s) : S \times A \times S \rightarrow \mathbb{R}$ is the Markovian reward function.
- $\gamma \in [0, 1]$ is the discount factor used to discount future rewards.

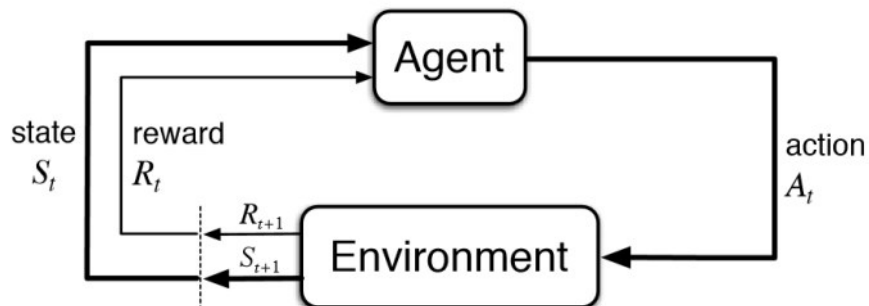


Figure 2.1: The RL framework from Sutton et al. (1998).

Intuitively, every timestep t the agent executes an action a_t based on the the current state s_t . This causes a transition in the environment according to the transition function P that updates s_t yielding a reward for the agent according to the reward function R .

Given a policy $\pi : A \times S \rightarrow [0, 1]$ that maps states to actions, the value V of state s is defined as the discounted sum of rewards:

$$V^\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t r_t \mid s_0 = s \right] \quad (2.1)$$

We also refer to the action-value function as the value or quality of taking action a at state s while following a policy π thereafter:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t \geq 1} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \quad (2.2)$$

We can now define the goal of RL as finding the optimal policy π^* that satisfies:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \quad (2.3)$$

Finally, we define the *advantage* function as the difference over the expected reward from taking action a :

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (2.4)$$

A core assumption of MDPs and vanilla RL in general is that rewards depend only on the current state and action taken, i.e., the past does no influence the decision-making process, which is called the *Markovian assumption*. Therefore, an MDP assumes that s contains all the relevant information from the past for the decision-making problem.

One of the core shortcomings of MDPs is their assumption of an agent with access to s . This is only true in problems where the agent is “omniscient” about the environment, i.e., it has constant access to all the relevant information in the environment. A good example of such kind of setting is chess. However, in most practical applications the agent will only have access to pieces of s . For example, in autonomous driving, it is not feasible to assume that the AI car will have flawless and constant

access to the exact location of every other vehicle, pedestrian, obstacle and any remaining concerning element on the road. A framework better designed for such environments is partial observable (p.o.) MDPs:

Definition 2.2 (POMDP). *A partially observable MDP is a tuple $\mathcal{M} = \langle S, A, P, R, Z, O, \gamma \rangle$ where*

- *S, A, P, R, γ are defined as in an MDP.*
- *Z is the set of observations $z, z', \dots$*
- *$O(s, a, s) : S \times A \times S \rightarrow Z$ is the observation function.*

In a POMDP, the agent only has access to the information provided by the environment through observation z . Since z_t alone might not provide enough context for the agent to find the optimal policy at a given timestep t , POMDPs lead often to non-Markovian settings in practice. Thus, in this setting policies typically map action-observation histories to actions $\pi : A \times (Z \times A)^* \rightarrow [0, 1]$. Various methods have been proposed in the literature to either explicitly estimate the probability distribution of the underlying Markov states or to stack past observations with recurrent neural networks (see Sec. 2.2.2) to implicitly reconstruct Markov states.

Another relevant variation of MDPs are the Markov Games, which are formulated for those environments where there are multiple learning agents:

Definition 2.3 (MG). *A Markov game is a tuple $\mathcal{G} = \langle N, S, U, P, R, \gamma \rangle$ where:*

- *S and γ are defined as in an MDP.*
- *N is the set of n agents.*
- *U is the action set where each A_i is the action set of agent $i \in N$. u_t refers to the joint action taken by all agents in timestep t .*
- *$P : S \times U \times S \rightarrow [0, 1]$ is the transition probability function that returns the probability of transitioning to a new state given the previous state and the actions taken by all the agents.*

- $R_i : S \times U \rightarrow \mathbb{R}$ is the reward function of agent i and $R = \{R_1, \dots, R_N\}$ is the set of reward functions.

MGs can be thought of as multi-agent extension of MDPs where actions are chosen and executed simultaneously, thus new states depend on the action taken by each single agent. MGs can also be extended to account for partial observability through dec-POMDPs (Oliehoek et al. 2016):

Definition 2.4 (dec-POMDP). A decentralised p.o. Markov Decision Process is a tuple $\mathcal{M} = \langle S, U, P, R, Z, O, \gamma \rangle$ where:

- S, N, U, P, R, γ are defined as in an MG.
- Z is defined as in a POMDP.
- $O(s, a, s) : S \times U \times S \rightarrow Z$ is the observation function.

Last, one important variant of MDPs are contextual MDPS (Hallak, Castro & Mannor 2015). This mathematical framework is specially used to formulate problems where agents need to master different environments that share a common structure:

Definition 2.5 (contextual MDP). A Contextual Markov Decision Process is a tuple $\mathcal{M} = \langle S, A, P, R, C, \gamma \rangle$ where:

- S, A, γ are defined as in an MDP.
- C is the context space where $c \in C$ is the current context.
- $P(s', c \mid s, c, a) : (S \times C) \times A \times (S \times C) \rightarrow [0, 1]$ is the transition function.
- $R((s, c), a, (s, c)) : (S \times C) \times A \times (S \times C) \rightarrow \mathbb{R}$ is the reward function.
- $P(s', c' \mid s, c, a) = 0$ if $c' \neq c$

Contextual MDPs are thought for episodic problems. Note that the last point in Definition 2.5 implies that the context remains fixed through an episode.

2.1.2 Approximating the Value Functions

In practice we do not have no access to the value function to estimate an effective policy from it, i.e., we do not know what future rewards will yield every action on every state given a policy. Nevertheless, there are multiple approaches for approximating value functions. One is *dynamic programming* (DP) (Sutton 1990), which involves iteratively estimating the value function using the Bellman equation, as outlined in Equation 2.1. However, the caveat with DP lies in its requirement of knowing the environment dynamics, thereby limiting its practicality.

Monte Carlo (MC) methods present an alternative paradigm (Barto & Duff 1993). These techniques dispense with the necessity of pre-known model dynamics. Instead, they construct an approximation of the value function by averaging over trajectories derived directly from the agent’s experiential data. Monte Carlo methods are especially effective in domains with delayed rewards (Silver et al. 2016).

Temporal difference (TD) (Tesauro et al. 1995), on the other hand, combine aspects of the two previous approaches. Similar to MC, they operate without demanding explicit knowledge of the environment model. Leveraging data from experience, TD methods continually update their value function estimate. Notably, akin to dynamic programming, TD capitalises on the recursive nature of the value function. It incorporates estimations from previous updates, employing them to bootstrap the ongoing approximation of the value function.

Our work employs algorithms based on TD since it is the most generalist approach and the standard solution in current DRL methods.

2.1.3 Policy Gradient

Policy gradient algorithms focus on the direct learning of a policy by maximising the probability of selecting actions that result in favorable outcomes. The policy, parameterised by θ , is optimised to maximise the expected total reward of a trajectory, as represented by the objective function:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} R(\tau) \quad (2.5)$$

where τ refers to a trajectory, i.e., a sequence of states and actions.

In the context of DRL, where neural network policies are prevalent, the optimisation of θ involves iterative gradient steps:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\pi_{\theta_t}) \quad (2.6)$$

in which $\alpha \in [0, 1]$ is a learning rate hyperparameter.

To derive the gradient for Equation 2.6, considering the MDP factorisation into transitions and expected actions from given states, the following expression is obtained (Peters & Schaal 2008, Sutton et al. 1999):

$$\nabla_{\theta} J(\pi_{\theta_t}) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi(a_t | s_t) R(\tau) \right] \quad (2.7)$$

However, a more refined attribution of credit is sought. Note that in Equation 2.7 each action is given the reward of the entire trajectory. Alternatively, many use the reward-to-go formulation:

$$\nabla_{\theta} J(\pi_{\theta_t}) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi(a_t | s_t) \sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}) \right] \quad (2.8)$$

Here, each action is credited with the cumulative reward from its occurrence until the end of the trajectory. Furthermore, to reduce variance, a baseline can be subtracted from the reward-to-go (Sutton et al. 1999):

$$\nabla_{\theta} J(\pi_{\theta_t}) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi(a_t | s_t) \left(\sum_{t'=t}^T R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t) \right) \right] \quad (2.9)$$

where $b(s_t)$ is typically the value function of the state $V^{\pi}(s_t)$. In this context, the thesis explores the estimation of the value function using a neural network, achieved by minimising the mean squared error (defined in Equation (2.10) below) between the model prediction and the true reward-to-go.

Equation 2.8 aims to enhance the probability of selecting actions that lead to improved performance relative to predicted returns. Moreover, we can also make use of the Q-function and have $b(s) = Q^{\pi}(s, a)$. Such methods are generally known as *actor-critic* algorithms where the critic assumes the form of the value function, evaluating how good an action is by subtracting the expected reward

from state from the reward obtained from the policy in that iteration (Sutton et al. 1998). Thus, we can utilise the learned Q-values to determine the *advantage* of taking an action in a specific state following Equation 2.4. This approach intertwines policy learning and value estimation, contributing to the versatility and effectiveness of actor-critic reinforcement learning algorithms (Mnih et al. 2016, Espeholt et al. 2018).

2.1.4 Model-Based RL

Up to this point, the methodologies discussed have been tailored for online learning from experience generated directly in the environment the agent is trying to solve (model-free), which out of simplicity we consider as an MDP for our current example. In contrast, model-based approaches aim to construct a model of the environment by approximating both the transition and reward functions, denoted as P and R , respectively. This model – commonly referred to as a *world model* – is often implemented as a deep neural network (see Section 2.2), with its parameters represented by ψ . The approximate dynamics and reward functions are denoted as P_ψ and R_ψ , respectively. Consequently, the world model introduces an “imaginary” MDP, denoted as $M_\psi = (S, A, P_\psi, R_\psi, \gamma)$.

The focus of this research lies in Dyna-style model-based reinforcement learning (Sutton & Singh 1994), where the training of a policy π_θ involves using latent or “imagined” transitions exclusively within the framework of M_ψ . Model-free algorithms can be employed to optimise the policy, and the training process can be parallelised on a single hardware accelerator since the simulator now operates as a neural network world model (Kurutach et al. 2018). In scenarios where learning occurs from low-dimensional states, a common approach involves using an ensemble of probabilistic dynamics models, initially introduced in Nix & Weigend (1994) and proven effective for learning world models (Chua et al. 2018, Janner et al. 2019). World models have also demonstrated efficacy in handling high-dimensional observations like pixels, often leveraging variational encoders (Kingma & Welling 2014). Pioneered by Ha & Schmidhuber (2018), subsequent works, including PlaNet (Hafner et al. 2019) and all the versions of Dreamer (Hafner et al. 2020, 2021, 2023), have expanded on this architecture. These approaches typically rely on a model with two main components: an encoder, mapping sensory inputs to latent codes, and a Recurrent State Space Model (RSSM), predicting future latent codes

based on actions and previous latent states. In Chapter 3 we discuss how these world models can be extended with a learning shield component to enhance safety.

2.1.5 Multi-agent Reinforcement Learning

One of the ultimate goals for autonomous agents is that they can effectively interact with both their environment and other agents. Multi-agent reinforcement learning (MARL) (Zhang et al. 2021) extends the framework of RL to environments where multiple agents interact simultaneously. Unlike vanilla reinforcement learning, which involves a single agent operating in an environment, MARL focuses on settings where the actions of each agent affect not only their own outcomes but also those of other agents. These settings are typically model according to if the agents have full or partial observability as Markov Games (Zachrisson 1964) or dec-POMDPS (Oliehoek et al. 2016) respectively. According to the intersection of the objectives between agents, we distinguish two different challenges:

- **Cooperative Games:** In cooperative MARL, agents work together towards a common goal. The success of one agent typically contributes to the success of others, and the aim is to maximise a shared reward function.
- **Competitive Games:** Contrarily, in competitive MARL, agents have opposing goals. The success of one agent usually comes at the expense of another. These are often modeled as *zero-sum games*, where the gain of one agent is exactly balanced by the loss of another.

Note that there can be a case where the objectives of some agents are orthogonal to those of other agents. However, those problems are more easily addressed not from a MARL perspective but as a single-agent RL problem in a non-stationary MDP (Kiran et al. 2021). Diving in multi-agent systems (MAS), one of the main challenges is that the optimal strategy depends also on the strategies of other agents. Thus, we use the concept of equilibrium from multi-agent systems (MAS), to refer to possible stable points that the agents may converge to. Two of the most common equilibrium (Wooldridge 2009) are:

- **Nash equilibrium:** Nash Equilibrium occurs when each player's strategy is optimal given the strategies of all other players in the game.
- **Correlated Equilibrium:** Unlike the Nash equilibrium, where each agent makes decisions independently, in a correlated equilibrium, agents may base their strategies on external signals from the state of the system. That is, in a correlated equilibrium agents do not assume the others are acting optimal, and try to infer their actions from the observations of the environment.

Last, another topic that is relevant for us is the distinction between centralised and decentralised MARL. *Decentralised MARL* refers to those methods that train agents to operate based on their local observations only (Tampuu et al. 2015). *Centralised MARL* involves making decisions based on the global state of the environment and actions of all agents. Centralised approaches are often more applicable in cooperative scenarios where a joint action-value can be optimised. This is typically done through Centralised Training for Decentralised Execution (CTDE), where agents are trained offline using centralised global information but execute in a decentralised manner online (Rashid et al. 2018, Mahajan et al. 2019).

2.2 Deep Learning

Deep learning (DL) encompasses a collection of techniques aimed at constructing parametric differentiable computation graphs and subsequently training them from data through gradient-based optimisation. This section aims to elucidate some pivotal components about the inner mechanisms in DL and the key elements utilised in this work. Specifically, we start introducing what are artificial neural networks and the inner mechanisms that make them function. Then, we give an overview of the different types of layers that we employ in our architectures. Lastly, we review the main types of activation functions that we use. For a more in depth review of the topics illustrated here we refer the reader to LeCun et al. (2015).

2.2.1 Artificial Neural Networks

A *deep learning* model, often referred to as an *artificial neural network*, or neural network (NN) for brevity, constitutes a network comprising of parameterised functional modules, typically organised in layers, facilitating computation from input to output layers. Specifically, this section starts introducing NNs and the key mechanism that make deep architectures work. Then we describe the different types of deep learning layers utilised through this work and finish with a brief overview of the main non-linear functions that we employ in these architectures to approximate functions of additional complexity.

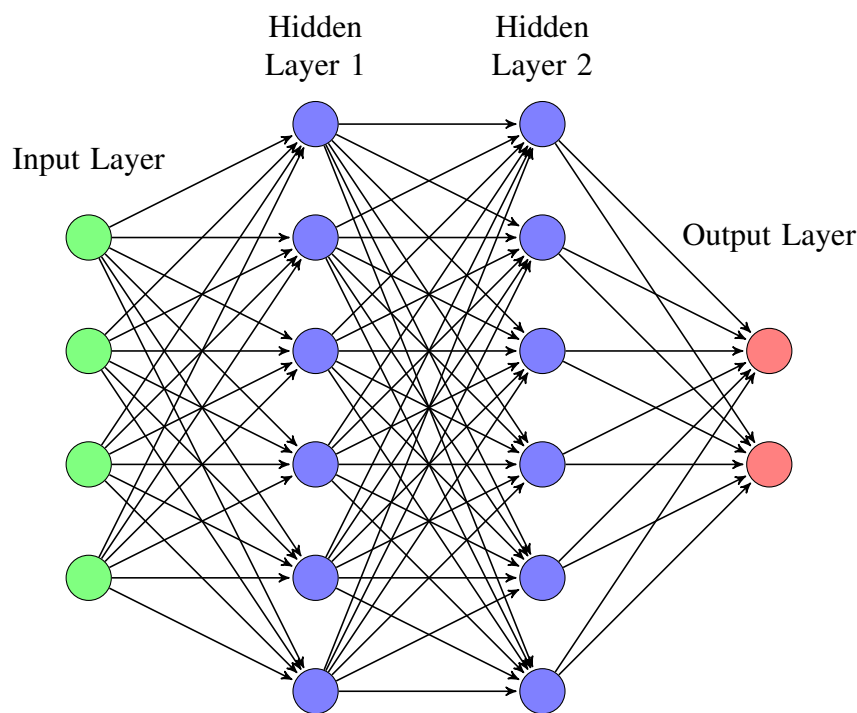


Figure 2.2: Illustration of the simplest type of NN, an MLP. This one has an input layer with four artificial neurons followed by two hidden layers of six neurons each that terminates in an output layer of two neurons.

Universal approximation theorem. The universal approximation theorem states that a neural network with at least one hidden layer can adeptly approximate any continuous function by adjusting the number of units (Cybenko 1989, Barron 1993). Moreover, the theorem highlights that expanding the number of hidden layers empowers networks to perform more intricate hierarchical computations. While deeper networks possess greater computational capability, they also pose challenges in terms of training complexity. The term “deep” in deep learning reflects the emphasis on this potent dimension

of depth, where the increasing depth of networks contributes to enhanced computational capacity but introduces heightened training difficulty.

Loss Function. As anticipated, NNs aim to minimise an objective or loss function. While there are many options to define such loss functions, here we introduce two common choices that we make use of in this work:

- *Mean Squared Error (MSE)* represents the Euclidean distance with the objective. This loss is easy to compute, stable and has analytic solution. However, because of the square it is sensible to outliers. MSE is commonly used for linear regression and as objective function for the critic, and is formulated as:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{n=1}^N (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2 \quad (2.10)$$

- *Cross entropy*, also known as log likelihood. It is a common choice for classification tasks. In our context is a popular option for autoencoders (see Section 2.2.2). A particular property of cross entropy is that it penalises heavily the predictions that are confident but wrong. Its mathematical formulation is as follows:

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{d} \log(\hat{\mathbf{y}}) + (1 - \mathbf{d}) \log(1 - \hat{\mathbf{y}}) \quad (2.11)$$

Backpropagation. Backpropagation serves as a cornerstone in deep learning (Rumelhart et al. 1986). It hinges on gradient-based optimisation methods, particularly gradient descent and its variations. To perform gradient descent, we do a forward pass through the network using a batch of inputs, generating outputs used to compute a loss. Subsequently, the backpropagation algorithm, or backprop, efficiently calculates the gradient of the loss applying the chain rule. This process iterates backward through every layer, starting from the last one.

Optimisers. Optimisers enhance vanilla gradient descent to facilitate convergence. Momentum, for instance, combines the current gradient with the previous update, leveraging inertia to overcome

noisy gradient oscillations and escape flat regions in the search space (Polyak 1964). Other techniques, incorporating second-order derivatives, yield more informed updates (Tieleman et al. 2012). Some optimisers require significant memory and compute resources (Tieleman et al. 2012, Kingma & Ba 2015), necessitating a trade-off between sample efficiency and processing time that is becoming increasingly important as the size of state-of-the-art (SOTA) models increases exponentially. Notable examples of optimisers include AdaGrad (Duchi et al. 2011), RMSProp (Tieleman et al. 2012) and Adam (Kingma & Ba 2015).

Batches. In theory, if the learning rate is small enough, gradient descent reduces the value of the loss function when the gradient is computed from the whole dataset (Goodfellow et al. 2016). However, using the dataset entirely in every training step is rarely feasible in any application of interest due to impractical memory requirements. Indeed, stochastic gradient descent is the mainstream approach, involving iterative gradient descent steps using smaller batches of data to approximate full batch gradient descent (Mnih et al. 2013). The use of batches not only aids with more manageable memory usage but also facilitates faster inference and backpropagation per input, leveraging the parallelisation capabilities of hardware and software.

2.2.2 Deep Layers

Within deep neural networks’s large architectures, layers serve as the foundational components. Their strategic stacking enhances network depth, consequently enhancing expressiveness (Goodfellow et al. 2016). Here we introduce the most common types of layers from existing literature that we employ in this work. Specifically, understanding the key concepts of these layers is essential to understand our contributions in Part II:

Fully connected layers. The simplest layer type is the fully connected layer, also known as *dense layer* or perceptron. It necessitates a single hyperparameter – the number of output units — and comprises two training parameters including a weight matrix W and a bias vector b . Given an input tensor X and an activation function f , the output Y is computed as $Y = f(W.X + b)$. Multilayer

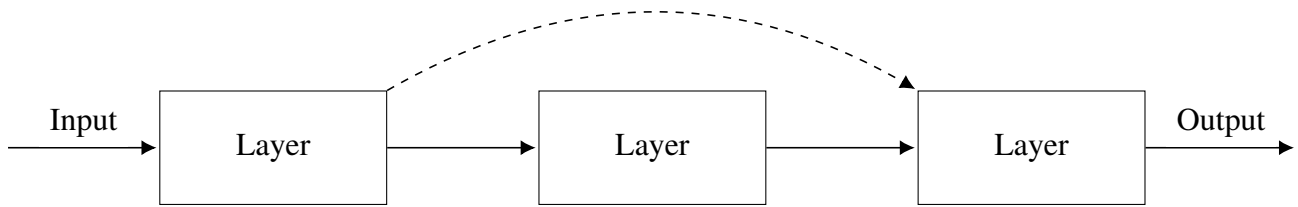


Figure 2.3: Depiction of a residual connection. The information from the residual path is typically incorporated to the standard flow of information through addition or concatenation.

perceptrons (MLPs) are essentially stacks of fully connected layers (Figure 2.2). For the sake of simplicity, we will refer to any number of fully connected layers as MLPs.

Residual connections. Deep neural networks often grapple with the *vanishing gradient* problem, where gradients diminish as depth grows, which is further exacerbated by certain activation functions squashing values and reducing gradients. One of the most popular solutions to overcome this issue are residual connections, also known as skip connection. Here the output of a layer is reused at later stages of the network as depicted in Figure 2.3. A prominent example is the residual neural network - known as ResNet-, where the input of a block of layers is added to the output of the same block, facilitating information retention and establishing shorter paths for gradients to flow through.

AutoEncoders. Not strictly a layer but rather a specific class of NNs. First introduced by Kramer (1991), autoencoders are designed for unsupervised learning of data encodings. Their primary objective is to acquire a lower-dimensional representation (encoding) of higher-dimensional data, commonly employed for dimensionality reduction. Figure 2.4 illustrates a standard autoencoder, where the fundamental components are:

- *Encoder*: This module is responsible for compressing the input data from the train-validate-test set into an encoded representation. Typically, this representation is significantly smaller in scale compared to the original input data.
- *Bottleneck*: Serving as a crucial component, the bottleneck houses the compressed knowledge representations. It is considered the most pivotal part of the network, encapsulating essential information from the input. This is the "output" we aim to utilise once training is finished.

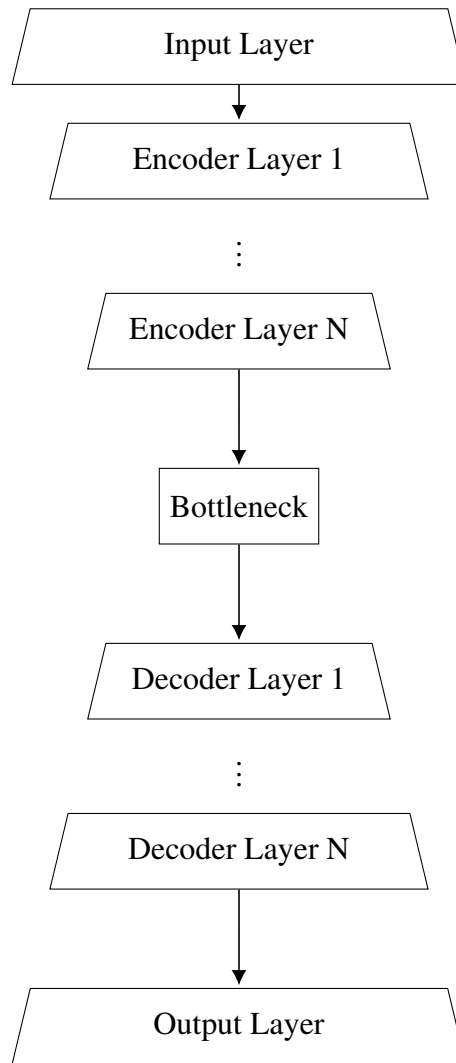


Figure 2.4: Illustration of an Autoencoder N layers for encoder and decoder, respectively.

- *Decoder*: The decoder module plays a vital role in “decompressing” the knowledge representations obtained from the bottleneck. It serves to generate the self-supervised objective since it reconstructs the input data back from its encoded form. Then, the output is subsequently compared with a ground truth for error correction.

Convolutional Layers. The concept of 2D convolutional layers loosely draws inspiration from the organisation of neurons in the visual cortex (LeCun et al. 1995). This layer applies a *kernel* with a receptive field over a 2D feature map, which is essentially a tensor with width, height, and depth. The depth corresponds to the number of channels, where grayscale images have one channel, and color RGB images have three channels for red, green, and blue. Convolution involves sliding a kernel over the input feature map to create an output feature map, establishing translation invariance through

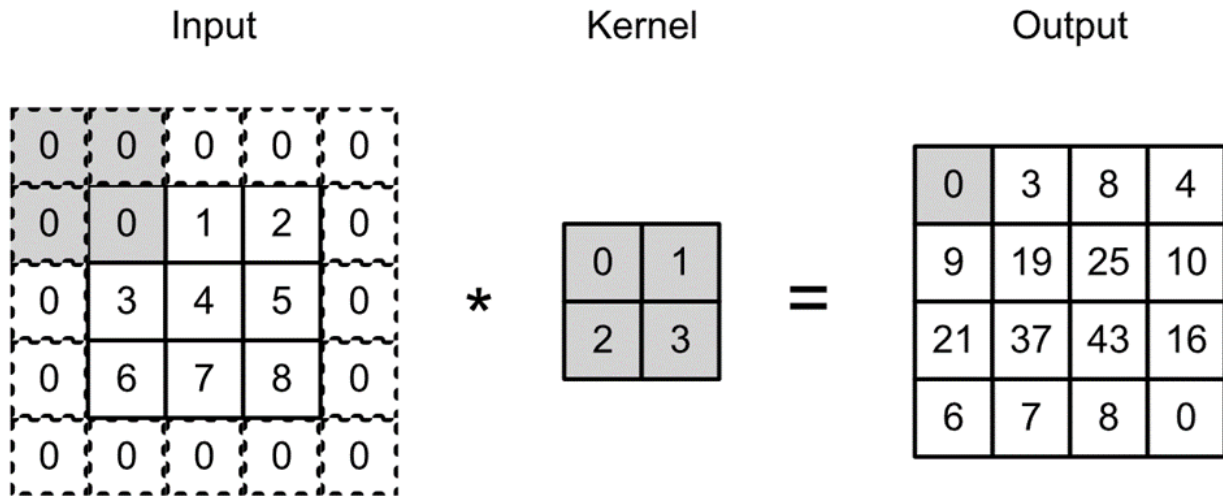


Figure 2.5: Example of a convolution with a 2x2 kernel with one channel. Note that stride is two since we move the kernel two positions for every operation. This example uses a padding of dimension 1 reflected by the zeros surrounding the input.

pattern matching. The kernel, with specified width, height, and depth, operates similarly to a fully connected layer in three dimensions, involving tensor weight multiplication and tensor bias addition. Besides the kernel size, another relevant feature of convolutional layers for our study in Chapter 6 is the *stride*, which measures the step-size of the kernel slid. Additional key hyperparameters for 2D convolutional layers include *padding*, and *dilation rate*. Figure 2.5 illustrates some of the key components of a convolution. After the last convolutional layer, the output is a tensor that we refer to as *visual embedding*. Moreover, downsampling can be achieved through pooling layers that aggregate neighbouring values, for example, with a max operator, to reduce the size of the feature maps. A convolutional neural network, often abbreviated ConvNet or CNN, consists of stacked convolutional layers, followed by flattening that gives us the visual embedding, and a variable number of fully connected layers. CNNs are traditionally employed within DRL literature to encode the agent’s observation when working with pixel-based inputs.

Recurrent Layers. Recurrent layers enable neural networks to process sequential input data while retaining information over time. These layers maintain an internal state across unroll steps, with a hyperparameter specifying the output size. The basic form of a recurrent neural network (RNN), i.e., an NN with at least one recurrent layer, involves taking both the current input and the previous output, concatenating them, and generating a new output using an internal dense layer. This process

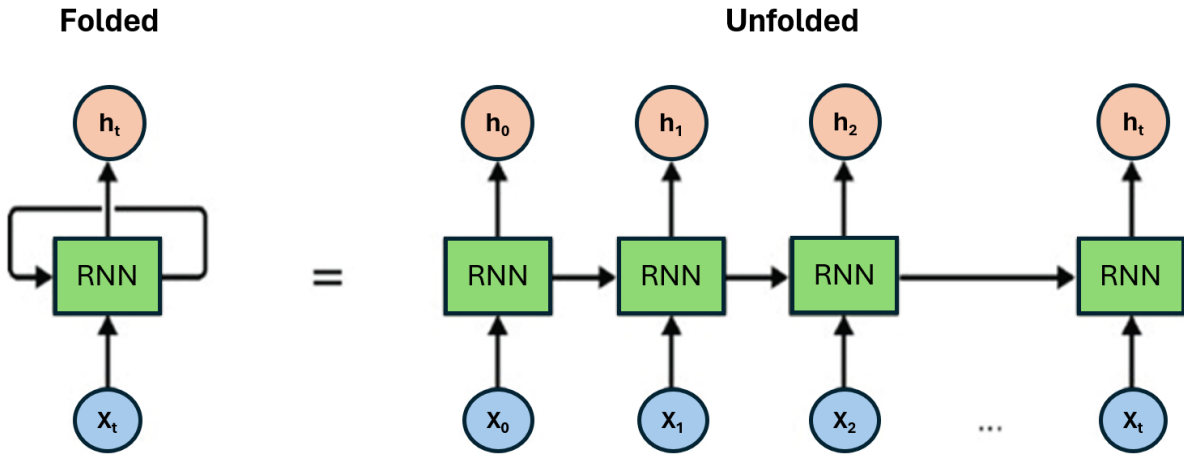


Figure 2.6: Illustration of how an RNN iterates through time.

of reusing the previous output to generate the next one typically continues through multiple unroll steps (see Figure 2.6). Note that an RNN differs from residual connections in that the additional information forwarded comes from a previous timestep, not from a previous layer. The sequence of inputs and outputs can serve various purposes, such as many-to-one (only the last output is sent to the next layer), one-to-many (the same input is repeated multiple times in input, and the many outputs are sent to the next layer), or many-to-many tasks (at every unroll step, a different input is provided, and the generated output is sent to the next layer). However, simple RNNs face challenges in retaining information over extended sequences, leading to vanishing gradients. To address this, long short-term memory (LSTM) (Hochreiter & Schmidhuber 1997) and gated recurrent unit (GRU) (Cho et al. 2014) layers have been introduced, incorporating gating mechanisms to enhance memory capacity and mitigate vanishing gradient issues. More recently, community has moved their focus toward the Structured State Space sequence model (S4) architecture and its variants (Gu et al. 2022). These networks bring additional features to the well known state space models (SSMs) (Chen & Brown 2013), effectively tackling significantly longer-term dependencies.

Relational Layers. Relational networks (RelNets) are a particular kind of neural network whose structure is explicitly designed for reasoning about relations (Kemp & Tenenbaum 2008). While works with DRL have been mainly focused on MLPs, RNNs and CNNs, RelNets are of particular interest in our context since they hold promise to improve generalisation in DRL (Santoro et al. 2018). Their central principle is to constrain the functional form of the neural network in the relational

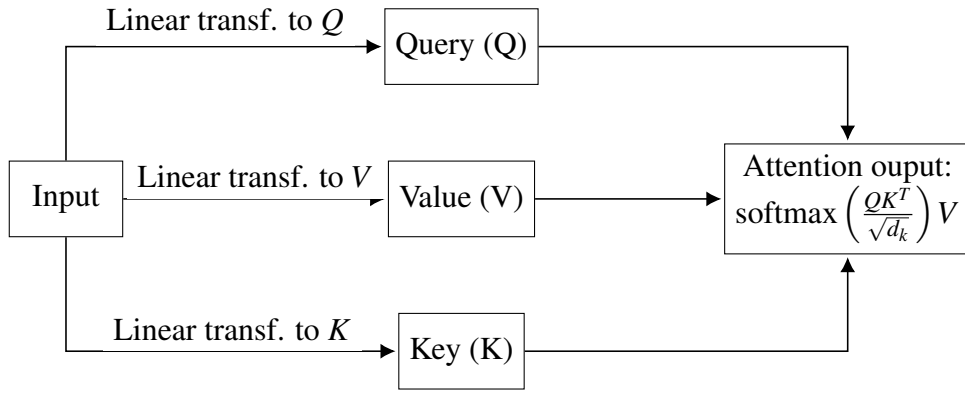


Figure 2.7: Illustration of a self-attention layer, i.e., an attention layer that obtains the relationship between the different pieces within the input.

module so that it captures the core common properties needed to reason about the relations between entities (e.g., objects) and their properties (e.g. color), see Santoro et al. (2017). Recently, relational networks have been enhanced with the inclusion of the widely-known key-value *attention* mechanism (Bahdanau et al. 2015). This method relies on the computation of attention scores that are used to decide in which portions of the input data the layer using this mechanism should focus on.

In its simplest form, a relational network is a function:

$$\text{RN}(o) = f_{\phi} \left(\sum_{i,j} g_{\theta} (o_i, o_j) \right) \quad (2.12)$$

where the input is a set of “objects” $Ob = \{ob_1, ob_2, \dots, ob_n\} \subseteq \mathbb{R}^m$, and f_{ϕ} and g_{θ} are functions with parameters ϕ and θ , respectively. For the networks studied in this work, these functions are implemented as MLPs. The output of g_{θ} is called a “relation” and infers if two objects are related.

Attention-based Layers. A key feature of the latest works on relational modules is the inclusion of content-based, key-value (attention for short) mechanisms (Zambaldi et al. 2018, Shanahan et al. 2020). This mechanism relies on the principle that, when deciding or planning, an agent does not need to devote equal attention to all subsystems at every time step, but only to those aspects that have a significant interaction with the agent (Bengio 2017). Attention involves the product of a query (also called read key) represented as a matrix Q , keys K (d -dimensional). The product of queries and keys -known as *attention score*- is normalised with a softmax function, and the result is applied to a matrix

of values V . The operation is:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (2.13)$$

where the output is a convex combination of the values in the rows of V . Hence, if the attention is focused on one element for a particular row, i.e., the softmax is saturated, this simply selects one of the objects and copies its value to the given row of the result. Figure 2.7 illustrates a self-attention layer.

An important variant of this attention mechanism is called *top-k attention* where the output values are only computed over the k inputs with the highest attention scores. This attention variant blocks all information coming from irrelevant parts of the input and it is specially relevant when having to train agents for different tasks or environments. For more details on attention in neural networks we refer to (Bahdanau et al. 2015, Vaswani et al. 2017).

2.2.3 Activation and Probabilistic functions

As a result of stacking multiple layers, deep neural networks of sufficient capacity are able to approximate functions of arbitrary complexity. To attain this capability, it is essential to incorporate non-linear activation functions between the linear layers described above. These are usually described as:

$$\mathbf{h}_i = g(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i) \quad (2.14)$$

Some of the most common choices to add non-linearity include:

Rectified Linear Units (ReLU). ReLU stands out as the most widely utilised activation function when adding non-linearity between different layers. it is denoted by:

$$g(x) = \max(0, x) \quad (2.15)$$

This function is particularly advantageous for optimisation, resembling a linear function for non-negative values. Variations of ReLU include leaky ReLU (Maas et al. 2013), which introduces a small fixed value instead of 0 in Equation 2.15, and parametric ReLU (He et al. 2015), where this small value is learned.

Logistic Sigmoid. The sigmoid activation function is defined as:

$$g(x) = \frac{1}{1 + \exp(-x)} \quad (2.16)$$

Sigmoid units have bounded values both above and below, saturating quickly for highly positive and negative inputs. However, their sensitivity is limited to a narrow range around 0. Currently their usage is mainly focused in normalised outputs.

Hyperbolic Tangent (Tanh). The hyperbolic tangent function, closely related to the sigmoid function, is expressed as:

$$\tanh(x) = 2\sigma(2x) - 1 \quad (2.17)$$

where $\sigma(x)$ represents the sigmoid function in Equation 2.16. Comparatively, hyperbolic tangent is often favored over sigmoid due to its passage through the origin (0, 0), resulting in behavior more akin to a linear function. Tanh is typically used as activation function in RNNs.

Softmax. An alternative approach used for representing probability distributions over the output values:

$$g(x) = \frac{\exp(x)}{\sum \exp(x_j)} \quad (2.18)$$

Softmax functions are usually applied to classification tasks, generating a probability distribution across potential labels.

2.3 Deep Reinforcement learning

The relevance of RL in the field of autonomous agents is due to the good performance that incorporating deep learning into RL (DRL) brings in complex environments with large state and action spaces. Here we describe fundamental algorithms used in this work.

2.3.1 Q-learning and DQL Variants

Q-learning (Christopher 1992) is an *off-policy* model-free RL algorithm. Off-policy refers to RL algorithms that are able to learn from data collected by any behavioral policy. Here we describe Q-learning and briefly introduce some algorithms derived from Q-learning that will be used in Chapter 4.

Q-learning works by initialising the Q-values of every state-action pair to some default value. Then, at every time-step, the algorithm uses a behaviour policy to execute an action a in the current state s , which leads to a new state s' and reward r returned by the environment as explained above. Given a learning rate α and a future-reward discount factor γ , the estimated Q-value ($\hat{Q}(s, a)$) is updated as follows:

$$\hat{Q}(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right] \quad (2.19)$$

Intuitively, Equation 2.19 means that Q-values, i.e., the “quality” of a given state-action pair, are updated according to the weighted sum of the old Q-values and the new sampled values according to the last trajectory.

This algorithm is guaranteed to converge to the optimal Q-values as long as every state-action pair is visited infinitely often (Christopher 1992). A method to fulfill this requirement is to set the behavioral policy to be ϵ -greedy on the target policy. That is, for each time-step, the behavior policy selects a random action with probability ϵ and the action with the highest Q-value (the one given by the target policy) with probability $1 - \epsilon$. Once convergence is achieved, one can simply obtain the optimal policy by always taking the action with the highest Q-value.

Deep Q-learning. Q-learning functions by using a table that stores every single state-action pair, which limits its application in large problems. A standard approach to overcome this issue is the Deep Q-learning (Mnih et al. 2015) (DQL). DQL use experience replay and deep neural networks parameterised by θ , where θ are the weights of the neural network to represent an action-value function for a given state. Intuitively, DQL employs neural networks as a function approximator of the table lookups from the original algorithm. In addition, DQL makes use of a replay memory buffer to store transition tuples of the form $\langle s, a, r, s' \rangle$. Function θ is learnt by sampling batches of transitions from the replay memory and minimising the following loss function:

$$\mathcal{L}(\theta) = \sum_{i=1}^M \left[\left(y_i^{\text{DQN}} - Q(s, a; \theta) \right)^2 \right] \quad (2.20)$$

where $y^{\text{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$ and θ^- are the parameters of a target network that are periodically updated (copied) from the parameters θ of the training network and kept fixed for a number of iterations. Note that even when DQL cannot guarantee to find the optimal policy, it is highly effective on large state spaces (Mnih et al. 2015).

Independent Deep-Q-Learning Independent Q-learning (IQL) (Tan 1993) is perhaps the most spread popular approach in early MARL. It basically decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. Tampuu et al. (2015) extended IQL with deep learning (I-DQL), and even while this approach does not address the non-stationarity problem introduced by the changing policies of the other agents, it nonetheless commonly serves as a strong benchmark for a wide range of MAS algorithms (Tampuu et al. 2017, Tang et al. 2018, Rashid et al. 2018).

LPOPL. Linear Temporal Logic Progression for Off-Policy Learning (LPOPL) is a Q-learning-based algorithm proposed by Toro Icarte et al. (2018). It focuses on teaching multiple tasks to a Reinforcement Learning (RL) agent. The core idea of LPOPL is to enable an RL agent to learn various tasks specified in co-safe Linear Temporal Logic (LTL), a formal language that we detailed in Section 2.4.2. LPOPL utilises LTL to specify tasks as patterns of properties or events that characterise the

successful execution of a task. One of the key contributions of LPOPL is that this algorithm exploits the structure of LTL to accelerate the learning process while keeping the guarantee of converging to an optimal policy. This is achieved by decomposing tasks into subtasks using LTL progression, which updates the task specification to reflect the parts of the task that have been accomplished or are yet to be achieved. For instance, in a complex task of building a hammer requiring multiple subskills: collect wood, iron, and then use a workbench, LPOPL decomposition allows reusing the learning of a those subtask for other tasks, e.g., collecting wood to build a different tool.

2.3.2 Advantage Actor Critic

Advantage Actor Critic (A2C) is a synchronous version of A3C (Mnih et al. 2016), it is an *on-policy* actor-critic approach that will be employed in the experiments of Chapters 6 and 7. On-policy means that the agent learns from data collected from the current policy, and it implies that once the policy network has been updated we need to discard all previous data. For this reason, on-policy methods are stable but inefficient. Nevertheless they serve well for many of the problems we explore in those chapters.

Algorithm 1 includes the pseudocode of A2C. Intuitively, training this algorithm is equivalent to train a critic network and then a policy network in a similar fashion to the REINFORCE algorithm (Williams 1992). A2C starts by collecting samples of the rewards obtained by the policy given the current observation o_t and policy network parameters θ . When the episode finishes we estimate the value error to update the critic parameters θ_v and improve value estimation. In parallel the policy network updates its parameters θ to increase the probability of sampling the observation- action pairs that yield higher advantage, where the advantage is calculated through Equation 2.4 by the critic network. The policy also includes an entropy regularisation term H in its objective to boost exploration with β_e being a hyperparameter to regulate the influence of such entropy.

Note that in this work we follow the common approach of simplifying the setting by using a single NN with different output heads to get both policy and value outputs (actor and critic). This has proved to improve the stability of the actor network in multiple settings (Raileanu & Fergus 2021).

Algorithm 1 Advantage actor-critic

```

1: // Assume parameter vectors  $\theta$  and  $\theta_v$ 
2: Initialize step counter  $t = 1$ 
3: repeat
4:   Reset gradients:  $\Delta\theta \leftarrow 0$  and  $\Delta\theta_v \leftarrow 0$ .
5:    $t_{start} = t$ 
6:   Get observation  $o_t$ 
7:   repeat
8:     Perform  $a_t$ , according to policy  $\pi(a_t|o_t; \theta)$ 
9:     Receive reward  $r_t$  and new state  $o_{t+1}$ 
10:     $t \leftarrow t + 1$ 
11:   until  $s_t$  is terminal or  $t - t_{start} == t_{episode}$ 
12:    $Q^\pi(o_t, a_t) = \begin{cases} 0 & \text{for terminal } s_t \\ V(o_t, \theta_v) & \text{for non-terminal } s_t \end{cases}$  // Bootstrap from last state
13:   for  $i \in \{t - 1, \dots, t_{start}\}$  do
14:     Accumulate gradients wrt  $\theta_v$ :
15:      $d\theta_v \leftarrow d\theta_v + \beta_v(Q^\pi(o_t, a_t) - V(o_i; \theta_v))(\partial V(o_i; \theta_v)/\partial \theta_v)$ 
16:     Accumulate gradients wrt  $\theta$ :
17:      $d\theta \leftarrow d\theta + \nabla_\theta \log \pi(a_i|o_i; \theta)(Q^\pi(o_t, a_t) - V(o_i; \theta_v)) + \beta_e \partial H(\pi(a_i|o_i; \theta))/\partial \theta$ 
18:   Perform update of  $\theta$  and  $\theta_v$  using  $\Delta\theta$  and  $\Delta\theta_v$  respectively.
19: until  $t > t_{max}$ 

```

2.4 Safety-Aware Reinforcement Learning

A major challenges of RL in real world applications is the existence of safety-critical constraints that are hard to balance with the reward maximising goals. Known examples of this problem are autonomous driving or high-cost robotics system (Kiran et al. 2021). In order to tackle this challenge, researchers developed new frameworks that not only aim to maximise long term rewards, but also to prevent damage (Alshiekh et al. 2018, Garcia & Fernández 2015a, Junges et al. 2016, Simão et al. 2021, Yang et al. 2023). Here we start briefly describing constrained reinforcement learning, a popular paradigm in related literature about RL in safety that we will be referring often to. Then, we dive into the main underlying concepts of approaches that combine formal methods and reinforcement learning that will be used through Parts II and I. Last, we will briefly introduce shielded reinforcement learning, another popular line of work in safe reinforcement learning and that we will be referring to in Chapter 3.

2.4.1 Constrained Reinforcement Learning

In constrained RL, the underlying decision making problem can be represented as a Constrained Markov Decision Problem (CMDP) (Altman 1993):

Definition 2.6 (CMDP). *A constrained MDP is a tuple $\mathcal{M} = \langle S, A, P, R, c, \zeta, \gamma \rangle$ where*

- S, A, P, R and γ are defined as in an MDP.
- $c_i : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ are the cost signals associated to every state-action, with $i = 1, \dots, m$ referring to m constraint objectives.
- ζ_i refers to the cost limit for the objective i .

Intuitively, an RL agent in a CMDP is solving a multi-objective problem, where we only accept the subset of policies that maximise future rewards while ensuring $c_i < \zeta_i$, for all $i \in [1, m]$. Thus, a simple approach to handle this kind of problems is through *Lagrange multipliers* (Rockafellar 1993). For instance, suppose a function $f(x, y)$ that needs to be maximised by the choice of x and y subject to the constraint that $g(x, y) \leq k$. The solution can be obtained through the Lagrangean function $L(x, y, \kappa) = f(x, y) + \kappa(k - g(x, y))$, where κ is the Lagrange multiplier. Abdolmaleki et al. (2018) employs this multiplier to automatically give different weights to the diverse objectives according to the current performance of the agent. However, constrained reinforcement learning is only applicable where we only care about safety at test time. When there are safety limitations at training time, neurosymbolic approaches like the ones described below are better suited for the task.

2.4.2 Formal Methods

Formal Methods (FM) are techniques of mathematical modelling for the specification, development and verification of both hardware and software systems. Since vanilla RL often fails at providing safety guarantees (Garcia & Fernández 2015b), there is a growing interest in bridging Formal Methods with Reinforcement Learning to bring the safety and verifiability features of the former to the later. Below we introduce key concepts from FM that are commonly used with RL.

Temporal Logic. When formally specifying a complex objective in RL, i.e., a multitask control goal, Temporal Logic (Huth & Ryan 2004) is very well suited and popular formal language. The core notion of Temporal Logic is that a formula is not statically true or false in a model, as takes place in propositional and predicate logic. Instead, models of Temporal Logic contain several states and a formula can be true in some states and false in others. Hence, the static notion of truth is replaced by a dynamic one, in which formulas may change their truth values as the system evolves from state to state. This dynamic notion makes temporal logic a suitable approach to express RL reward functions regarding the true or false state of a given formula.

LTL. Linear-Time Temporal Logic (*LTL*) is an expressive and well-studied temporal language to specify properties of both single- and multi-agent systems (Pnueli 1977). This language is many times the default choice in RL with temporal logic (RL-TF) due to its relative simplicity and its wide use in the literature of RL with temporal logic (Schillinger et al. 2019, Muniraj et al. 2018, Hatanaka et al. 2023). Formulae in *LTL* are built from a set $\mathcal{AP}$ of atoms, by using Boolean and temporal operators as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathbf{U}\varphi_2 \quad (2.21)$$

where $p \in \mathcal{AP}$, $\bigcirc$ is the *next* operator, and $\mathbf{U}$ is the *until* operator. We use the standard abbreviations: *eventually* $\diamond\varphi \equiv \top \mathbf{U}\varphi$; *always* $\Box\varphi \equiv \neg\diamond\neg\varphi$; *weak next* $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$ (when interpreted over finite traces $\neg\bigcirc\varphi \neq \bigcirc\neg\varphi$); and *Last* $\equiv \bullet\text{false}$ denotes the end of the trace (if finite).

The semantics of *LTL* formulae over $\mathcal{AP}$ are defined over infinite sequences $\sigma = \langle \sigma_0, \sigma_1, \sigma_2, \dots \rangle \in (2^{\mathcal{AP}})^\omega \cup (2^{\mathcal{AP}})^+$, where each σ_i is a truth assignment to each atom in $\mathcal{AP}$. By $p \in \sigma_i$, for atom $p \in \mathcal{AP}$, we mean that p is true in σ_i . The length of a sequence σ is denoted as $|\sigma|$, with $|\sigma| = \omega$ if σ is infinite. For $i \leq |\sigma|$, let $\sigma_{\geq i}$ be the suffix $\sigma_i, \sigma_{i+1}, \dots$ of σ starting at σ_i and $\sigma_{\leq i}$ its prefix $\sigma_0, \dots, \sigma_i$.

We can now define formally when a sequence σ *satisfies* an *LTL* formula φ at time $i \geq 0$, denoted by $\langle \sigma, i \rangle \models \varphi$, as follows (clauses for Boolean connective are immediate and thus omitted):

$$\begin{aligned} \langle \sigma, i \rangle \models p & \quad \text{iff } p \in \sigma_i \\ \langle \sigma, i \rangle \models \bigcirc\varphi & \quad \text{iff } |\sigma| \geq i + 1 \text{ and } \langle \sigma, i + 1 \rangle \models \varphi \end{aligned}$$

$\langle \sigma, i \rangle \models \varphi_1 \text{U} \varphi_2$ iff for some $j, i \leq j \leq |\sigma|$ and $\langle \sigma, j \rangle \models \varphi_2$, and for all $k, i \leq k < j$ implies $\langle \sigma, k \rangle \models \varphi_1$

A sequence σ is said to satisfy or model φ iff $\langle \sigma, 0 \rangle \models \varphi$. Notice that, whenever σ is infinite, we obtain the standard semantics for *LTL*.

There are two variants of *LTL* that are also relevant to our work in Chapter 4. Both are motivated by the fact that, in RL, agents are typically trained over finite episodes, but they tackle this feature in different way. The first variant is *LTL over finite traces* (*LTL_f*), which is standard *LTL* interpreted over finite traces only, instead of infinite ones (De Giacomo & Vardi 2013). The second variant is *co-safe LTL* (Kupferman & Vardi 2001), which – like *LTL* – is interpreted over infinite traces, but its syntax is restricted as follows:

$$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \text{U} \varphi_2 \quad (2.22)$$

The restricted syntax of *co-safe LTL* is meant to guarantee that formulas, if true, are true after a finite number of steps.

Model Checking. Model Checking is a technique commonly used in the RL with formal methods literature. It is a verification method from FM that explores all possible system states in a exhaustive manner (Baier & Katoen 2008). Traditionally, in an RL setting, model checking would refer to a software that receives a Markov chain - and MDP conditioned on the policy π to be evaluated- and examines all possible trajectories in a systematic manner. In this way, it can be shown that a given policy truly satisfies a certain property. This, makes model checking a useful technique to guarantee safety and is typically exploited by shield-based approaches that we describe below.

2.4.3 Shielded Reinforcement Learning

Shielded reinforcement learning refers to a family works that include a safety proxy between the agents and their environments when taking actions as depicted in Figure 2.8. Intuitively, shielded RL typically starts by formally defining what is acceptable or not (the *task*) that our policy does, and then

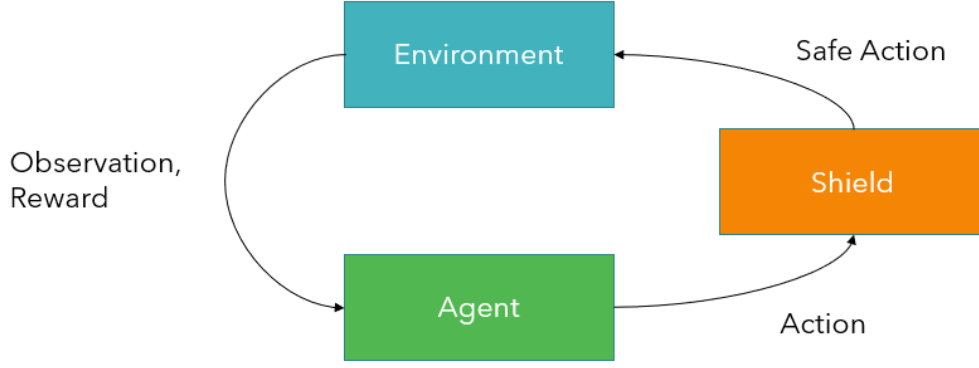


Figure 2.8: The Shielded RL framework

define high-level mechanisms to impede the agent from violating that task, i.e., (the shield), which for instance can be achieved with model checking.

More specifically, the *classical* formulation of shielding in RL is given by Alshiekh et al. (2018). It assumes access to two ingredients: an LTL safety specification and *abstraction* (an MDP model of the environment that captures the aspects of the environment relevant for planning ahead with respect to the safety specification). These ingredients are used to construct a formally verifiable reactive system that monitors the agent’s actions, overriding those which lead to violation states. One important limitation of early formulations of safety shielding is that they are impractical in large state spaces or continual learning settings since they require to check infinitely long trajectories. This motivated Giacobbe et al. (2021a) to define *bounded prescience shielding (BPS)*. For an arbitrary MDP with states $\mathcal{S}$ and actions $\mathcal{A}$, a *bounded trajectory* ρ of length H is a sequence of states and actions $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} s_n$ comprised of no more than H states and with the final state s_n either being a terminal state or $n = H - 1$. We can say a bounded trajectory ρ of length H satisfies *H-bounded safety* with respect to safety specification ϕ , written $S_H(\rho, \phi)$, if and only if for all $s_i \in \rho$, $L_{\mathcal{E}}^{\phi}(s_i) = \text{safe}$.

BPS allows shielding for a wider range of applications. That includes novel BPS model-based RL algorithms that we developed in works out of the scope of this thesis, we refer interested readers to He et al. (2022) for further detail.

Chapter 3

Related Literature

In this thesis we focus on improving generalisation and policy exploration in deep reinforcement learning, with special attention to safety and formal methods in RL. This chapter explores related works to those lines of research. Specifically, we start with Section 3.1 providing a general overview of safety-aware reinforcement learning literature through two main lines of work: constrained reinforcement learning (Section 3.1.1) and shielded reinforcement learning (Section 3.1.2). Later, we focus on research that is closer to our particular contributions. Specifically, we start with Section 3.2 reviewing works on systematic generalisation and modular networks, these pieces of literature are related to the topics studied in Chapters 6 and 7. Section 3.4 explores the subject of policy diversity (Section 3.4.1) and universal estimators (Section 3.4.2), both topics are foundational pieces of our work in Chapter 8. Lastly, Section 3.3 includes literature that combines formal methods and reinforcement learning, which is linked to the research in Part I (Chapters 4 and 5) and the neuro-symbolic frameworks in Part II (Chapters 6 and 7).

3.1 Safety-Aware Reinforcement Learning

Our work targets robust decision-making through reinforcement learning where agents need to attend to multiple goals and constraints. Thus, we are interested on algorithms that aim not only at reward

maximisation but also satisfying user-imposed constraints. Within safe RL literature we find two main lines of work: constrained reinforcement learning (Section 3.1.1) and shielded reinforcement learning (Section 3.1.2). We start this section with a brief overview of these two lines of work. However, as we will see through our presentation of these topics, constrained RL and shielded RL have fundamental limitations that make their solutions fragile when encountering distributions swifts deployments phase. This underlying weakness grounds our focus later in the section on literature about generalisation (Section 3.2), neuro-symbolic frameworks (Section 3.3), policy diversity (Section 3.4.1) and universal estimators (Section 3.4.2).

3.1.1 Constrained Reinforcement Learning

Constrained reinforcement learning addresses safety concerns by preventing agents from exploring potentially dangerous behaviors (Altman 1999, Uchibe & Doya 2007, Achiam et al. 2017, Marchesini et al. 2022). This is achieved with the addition of a cost function that needs to be minimised (Achiam et al. 2017). While the reward function encourages certain behaviors, the cost function is designed to penalise unwanted actions. According to the requisites of the problem, the user can establish limits to the cost function. Thus, constrained RL is a constrained optimisation problem (Borkar 2005), where the agent must only explore a sub-space of all available policies. Techniques developed in this fashion include constrained policy optimisation (CPO) (Achiam et al. 2017), which focuses on safe policy improvement, and safety-oriented search (SOS) (Marchesini et al. 2022) that adds a genetic algorithm phase to the training process. Another popular strategy, Lagrangian Proximal Policy Optimisation (L-PPO) (Ray et al. 2019), utilises the Lagrangian dual relaxation technique for constrained optimisation challenges while building on the benefits of PPO (Schulman et al. 2017) one of the most popular algorithms in RL.

Constrained RL has, however, two major limitations. The first one, also shared with other safety-aware methods, is the exploration-exploitation dilemma (Efroni et al. 2020). Constrained RL discourages policies as soon as they violate the constraint limits, which may have the undesired effect of discarding optimal policies that simply need further experience and improved control skills from the agent to satisfy those constraints. The second limitation is that constrained RL only provides a

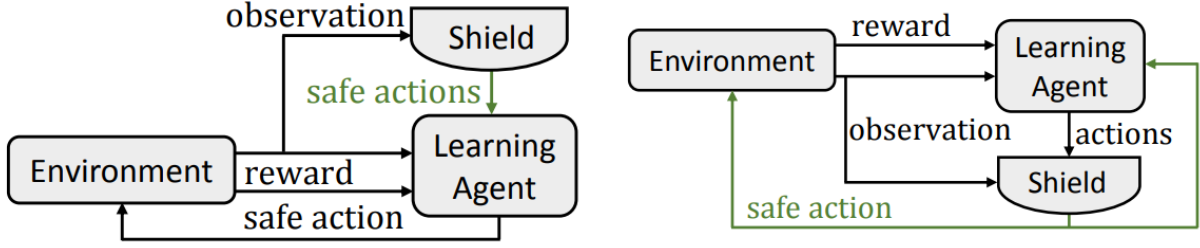


Figure 3.1: Types of shielding as first presented by Alshiekh et al. (2018). Left is an example of preemitive shielding. Right, is post-posed shielding.

probabilistic estimation of risk across an entire trajectory without ensuring safety at every individual state (Achiam et al. 2017, Liu et al. 2021, Corsi et al. 2023). This second limitation one of the main motivations of the literature on shielded RL that we detail below.

3.1.2 Shielded Reinforcement Learning

Most methods for integrating formal techniques into safe reinforcement learning (RL) follow a common strategy: they articulate requirements using formal logic, derive monitoring conditions from these requirements, and limit the RL agent’s operations to a portion of the state space that is verified as safe (Alshiekh et al. 2018, Fulton & Platzer 2018, 2019, Anderson et al. 2020, Carr et al. 2023, Hsu et al. 2023). *Shielding* is a notable method within this paradigm. Given a set of formal specifications defined in a formal language such as temporal logic, a *safety automaton* can be synthesised such that a trace will only be accepted by the automaton if the trace does not contain any unsafe states (Kupferman & Vardi 2001). Given a safety automaton and a suitable *abstraction* of the environment that captures the features relevant for planning with respect to the safety specification, a shield can be synthesised (El Mqirmi et al. 2021). The shield operates by preventing the agent from entering unsafe states, either by restricting the agent’s available actions before they are chosen, known as *preemptive* shields, or by modifying potentially dangerous actions after they are selected, *post-posed* shields, (Alshiekh et al. 2018). Figure 3.1 depicts the two approaches.. Shields are designed to be both effective at providing safety and minimally intrusive when interfering with the RL policy.

However, those earlier shielding techniques overlook the agent’s policy, potentially restricting exploration more than necessary. The REVEL (Reinforcement Learning with Formally Verified Ex-

ploration) framework addresses this by creating more tailored shields that enable greater exploration freedom (Anderson et al. 2020). This approach starts with a policy protected by a shield and then trains a neural network (NN) to mimic the shield’s behavior, ensuring safety by only deploying the shield during training. The NN’s policy is then refined, while the shield ensures ongoing safety, and the shield is updated through imitation learning to minimise divergence with the policy.

One shortcoming of the approaches described above is the necessity for a full abstracted representation of the environment with all the relevant information about the safety requirements. *Bounded prescience* (Giacobbe et al. 2021b) relaxes this constraint by considering a fixed future horizon at the expense of assuming access to the world’s simulator, e.g., a game engine. Thus, an agent that is safe under bounded prescience is said to satisfy *bounded safety*, i.e., it is safe for h steps, where h is the horizon limit. Despite its effectiveness, the bounded prescience approach faces limitations both in large state spaces and as horizon expands, particularly in scenarios where unsafe conditions are unavoidable after a certain point. This can be exemplified by a car falling off a cliff where the impending crash takes longer to happen than what bounded prescience can handle. Moreover, the methods described above rely on extensive prior knowledge, which may not be accessible in many real-world applications. We relaxed this issues in our work in He et al. (2022), where we expanded Dreamer (see Section 2.1.4) to incorporate a shield within the learned model to achieve *approximate bounded prescience shielding* (ABPS). A distinct feature of our solution is that the shield is not hand-crafted but learned, relying only on the assumption that the reward function will provide negative feedback when bad states are encountered. This allowed us to tackle setting where there is no access to a pre-existing world simulator -which applies to most real-world scenarios-. Furthermore, the approximate model of ABPS allows for computing resources to be allocated in the trajectories the agent is more likely to choose, allowing the user to compute longer horizons. This was further extended in Goodall & Belardinelli (2023) which employs critics to estimate unsafety beyond the horizon.

While shielding is a promising approach that we also explored in El Mqirmi et al. (2021) and He et al. (2022), it does not tackle fundamental issues in safety-aware reinforcement learning also present in other lines of work such as constrained reinforcement learning. Specifically, imposing safety constraints negatively impacts the robustness and generalisation capabilities of the agents, and severely

limits exploration (He et al. 2022, Odriozola-Olalde et al. 2023, Zhao et al. 2023). Thus, this thesis is devoted to find solutions to these core challenges.

3.2 Systematic Generalisation in Deep Reinforcement Learning

Generalisation in deep reinforcement learning is an active line of research since current algorithms still struggle to exhibit reliable behaviors in unseen scenarios (Kirk et al. 2023). Since the literature on generalisation is wide, we devote this section to the works exploring systematic generalisation in RL, which is the form of generalisation that we target in Chapters 6 and 7 and modular NN architectures, which is connected to our contributions in Chapter 7.

Systematic Generalisation. Systematic generalisation – also known as compositional generalisation (Lake 2019) – pertains the human ability of compositional learning, that is, the algebraic capacity to understand and execute novel utterances by combining already known primitives (Fodor & Pylyshyn 1988, Chomsky & Lightfoot 2002). For instance, once a human agent understands the instruction “get *wood*” and the meaning of *iron*, they will understand the task “get *iron*”. This is a desirable feature for a computational model, as it suggests that once the model is able to understand the components of a task, it should be able to satisfy tasks with the same components in possibly different combinations.

The struggle of neural-based agents to learn systematically and generalise beyond the training environment is a recurrent point of debate in machine learning (ML) (Bahdanau et al. 2019, Lake 2019). In the field of autonomous agents that follow human-given instructions, research on neuro-symbolic frameworks is gaining momentum (Camacho et al. 2019, Jothimurugan et al. 2019, Simão et al. 2021, den Hengst et al. 2022). Works along this line typically rely on structured or formal languages, which offer several desirable properties for reinforcement learning (RL) including unambiguous semantics, and compact compositional syntax. However, most of these works have either focused on solving training tasks (Icarte et al. 2018, Hasanbeig et al. 2021, Hammond et al. 2021) or rely on training from scratch new policy networks when tackling novel instructions (Kuo et al. 2020, Icarte et al.

2019), with its associated computational burden.

Focusing on relevant specific works in the field, we start with Oh et al. (2017), which presents a framework that relies on hierarchical RL and task decomposition of instructions in English to enhance generalisation in DRL. Later work from Mao et al. (2019) introduces a neuro-symbolic concept learner that jointly learns visual concepts, words, and semantic parsing of sentences from natural supervision. Yu et al. (2018) present a model that reports strong generalisation in a 2D environment that is aimed at language understanding with positive instructions. Closer to our line of work, there is growing literature (see e.g., Smolensky (1988), Bahdanau et al. (2019), Lake (2019)) focused on finding the right drivers enabling systematic learning from natural language instructions, e.g., fixed perspectives and varied training data. Recent work (Hill et al. 2020) suggests that systematic generalisation is not a binary question, but an *emergent* property of agents interacting with a situated environment (McClelland et al. 2010) and explores the drivers that enable agents generalise an abstract operator such as negation. More recently, Lake & Baroni (2023) shows how meta-learning can also be an important factor towards achieving human-level systematic generalisation. In Chapter 6, we expand the state of this line of works by first, providing evidence that systematic learning can emerge with logic operators from as few examples as with positive instructions and, second, by giving evidence that the CNN architecture is a key driver –that previous research was oblivious about– towards generalisation.

Modular Architectures. Chapter 7 has also links to modular deep learning architectures. These configurations hold promise in improving generalisation through its natural compositionality (Jothimurugan et al. 2019). Examples of modular frameworks include DRL agents that combine general control policies, where a global controller is able to combine with different lower-level modules in series of locomotion tasks (Huang et al. 2020). Closer to our work, Karkus et al. (2020) propose a framework that makes use of a perception and a planner networks to select the most suitable human-designed instruction for each time step to be forwarded to a controller network. This differs from our proposed framework in Chapter 7, where we induce a latent goal representation in deep learning networks that are learned from scratch in an unsupervised fashion, without using any predefined human task as embedding of the goal.

3.3 Neuro-Symbolic Reinforcement Learning

While deep reinforcement learning can exhibit some form of systematic generalisation, there are form of generalisation to which symbolic methods are better suited (Liu et al. 2023). Literature that combines formal methods and reinforcement learning is also close to our contributions in Parts II and I. One example is Alshiekh et al. (2018), that employ model checking to build a shielding system that is meant to prevent a learning agent from taking dangerous actions. In Mason et al. (2017) verification techniques are applied to abstract MDPs to check abstracted policies learned with RL. Related works (see, e.g., Zhou & Li (2018), Wen et al. (2017)) pursue similar verification goals in the sub-field of inverse RL, where reward functions are initially unknown and Formal Methods enable some degree of safety on the extracted rewards. Closer to our contribution in Chapter 4, Muniraj et al. (2018) tackles MARL with rewards expressed in temporal logic. Specifically, they enforce signal temporal logic (STL) specifications in a mini-max game extension of the popular decentralised I-DQN algorithm (see Section 2.3.1) for adversarial settings. Our approach in Chapter 4 differs under two key aspects. First, we provide a formal definition on how to use formal methods in order train multi-agent reinforcement learning policies to solve non-Markovian setting, which is not restricted to a specific temporal logic, such as STL. Second, our work is meant to cover a wide range of multi-agent scenarios, not only adversarial as in Muniraj et al. (2018).

Our work has also strong links with literature about autonomous agents that solve multi-task goals expressed in formal language. Examples include following instructions expressed as TL formulae by using logic progression and RL (Littman et al. 2017, Andreas et al. 2017), tackling continuous-action environments (Yuan et al. 2019), multi-agent systems (Hammond et al. 2021) and studies on the applicability of different TL languages (Camacho et al. 2019). Complex temporal formulae can also be easily decomposed by using other techniques, including finite state automata (De Giacomo et al. 2019, Icarte et al. 2018) and progression (Toro Icarte et al. 2018). More recent work has further exploited automata techniques, combining hierarchical approaches and reward machines – a particular form of automata– (Furelos-Blanco et al. 2021) or high-level planning with low-level RL solutions (Jothimurugan et al. 2021). These works focus on solving non-Markovian models by extending the state space in a minimal form (to minimally impact the state space) so that the RL agent learns from an equiv-

alent Markovian model, where an optimal behavior is also optimal in the original system (Bacchus et al. 1996). A drawback of the mentioned literature is their exclusive focus solving compositions of training tasks. For this reason Kuo et al. (2020) targets generalisation of temporal logic instructions with DRL at the expense of requiring an additional neural network for each new symbol or object encountered. Our work in Chapter 6 advances the state of the art by presenting a framework that can execute zero-shot TL formulae while relying on a single neural network. Later research that builds on our contributions in that chapter (Vaezipoor et al. 2021) tackles a similar problem in non-visual scenarios by pretraining a neural network to learn to interpret LTL instructions in navigation-based environments. Our work in Chapter 7 further advances the state-of-the-art in agents following formal instructions by presenting a novel network architecture that improves the performance of DRL algorithms executing zero-shot instructions in out-of-distribution environments.

3.4 Policy Exploration

The last body of related literature that we explore in this chapter is linked to our contributions in Chapter 8. That work targets strengthening robustness and exploration in constrained setting. Our contributions in Chapter 8 builds on top of two main bodies of research: diversity learning and universal function approximators. Hence, we organise this section accordingly.

3.4.1 Policy Diversity

Fostering diverse behaviors in autonomous agents is gaining attention in the RL community due to the growing evidence of the benefits that diversity learning brings to several core RL problems. Such benefits include generalisation (Eysenbach et al. 2018), exploration in single (Gehring et al. 2021) and multi-agent settings (Czarnecki et al. 2020), adaptation (Cully et al. 2015), and creativity (Zahavy et al. 2024). Moreover, policy diversity already had a central role in RL breakthroughs such as Vinyals et al. (2019), where self-training a league of diverse agents was key to achieve master level. These techniques have also become predominant in areas such as automated curriculum learning and unsupervised environment design (Wang et al. 2019, Dennis et al. 2020, Jiang et al. 2021,

Parker-Holder et al. 2022), generating frameworks that explore the different weaknesses of the training agent to strengthen them in an automated fashion.

Quality-Diversity (QD) optimisation is predominantly a family of evolutionary algorithms that aim at generating large collections of solutions that are both diverse and high performing (Pugh et al. 2016, Grillotti & Cully 2022). In QD, diversity is measured through a descriptor space defined by the user (Cully et al. 2015). Some of the main approaches in QD include MAP-Elites (Mouret & Clune 2015, Flageat et al. 2023), novelty search with local competition (Lehman & Stanley 2011) and more recently those that combine RL with QD (Nilsson & Cully 2021, Faldor et al. 2023, Tjanaka et al. 2022). Finally, a distinguishing characteristic of QD is their reliance on genetic algorithms, which can cause slow convergence in challenging environments (Pierrot et al. 2022). For this reason we focus on lines of work that foster diversity through the means of reward maximisation.

One of the central precursors in our line of work is Diversity Optimisation Maintaining Near Optimality (DOMiNO) (Zahavy et al. 2023). This algorithm employs reinforcement learning to maximise extrinsic and intrinsic diversity rewards, which are combined with Lagrange multipliers and Van der Waals (VDW) forces to balance the quality-diversity trade-off. DOMiNO has two variants: one bases its diversity objective on the average of input features which grants stability at the cost of storing a moving average for every task – limiting scalability. The second variant uses the critic to estimate successor features (SFs), which solves the scalability issue at the cost of learning instability due to added leaning target. Intuitively, such limitations prevent DOMiNO from being effective at discovering diverse behaviors in CMDPs. Further, the Lagrange multipliers and VDW forces can limit diversity in complex environments where creative behaviours need significant exploration before providing satisfying returns (Vinyals et al. 2017). Such limitations become more evident in complex physically-accurate environments such as GranTurismo (see Section 8.3). Our algorithm is instead designed to tackle those shortcomings.

3.4.2 Universal Estimators

The other central line of literature for our work in Chapter 8 pivots around *universal function approximators* (Schaul et al. 2015). This body of research factors the value estimates separating states from tasks (Schaul et al. 2015) or policies (Ma et al. 2020). Borsa et al. (2019) improve the formalisation of successor features by conditioning their estimation on task encodings and, as in Barreto et al. (2017), they exploit the General Policy Improvement (GPI) algorithm to show efficient behavior generation. Still, approaches based on SFs are brittle in complex domains (Carvalho et al. 2021, 2023), requiring complex network architectures to work effectively (Mittal et al. 2020, León et al. 2022b). Mozifian et al. (2021) attempt transferring learning of SFs to continuous domains by combining them with SAC Haarnoja et al. (2018), but fail to identify key elements that we study here. Chapter 8 combines the strengths of diversity with learning universal approximators for both SFs and discounted values while incorporating novel mechanisms providing a better optimisation objective for SFs than the one employed in Universal Successor Features Approximators (USFA) (see Appendix C.2).

Part I

Extending Non-Markovian Formalism to Complex Settings

Chapter 4

Solving Complex Non-Markovian tasks through MARL

The past do have a role in the future. Chapter based on León & Belardinelli (2020)

Through this work we highlighted that reinforcement learning (RL) is increasingly becoming a dominant paradigm for training autonomous agents in unknown and high-dimensional scenarios (Mnih et al. 2015, Schulman et al. 2017, Wurman et al. 2022). In Chapter 3 we reviewed work that combine RL with temporal logic (De Giacomo et al. 2018, Toro Icarte et al. 2018) and hold considerable promise to help learning agents tackle several tasks as a single specification, including safety constraints. Safety plays a major role in many real-world scenarios, such as autonomous driving (Shalev-Shwartz et al. 2016), robots (Junges et al. 2016) or network packet delivery (Ye et al. 2015). Additionally, temporal logic offers a simple but efficient way to use RL algorithms to find optimal policies in non-Markovian settings (Bacchus et al. 1996, Brafman et al. 2018, De Giacomo et al. 2018).

These real-world scenarios, in which agents have to work in an environment where there are other learning agents, are naturally modelled as a multi-agent system (MAS). Unfortunately, MAS have several challenges on their own, such as scalability with the number of agents as well as issues pertaining to non-stationarity (Canese et al. 2021), i.e, in MAS probabilities attached to transitions do

not depend solely on the single agent's actions and the corresponding transition probabilities.

This dependency on other agents' policies means that the notion of optimality, central to MDPs, needs to be encapsulated in a sort of equilibrium to solve the optimisation problem. In this context, Nash equilibrium (Hu et al. 1998), where each agent's policy is the best response to the others', is one of the most wide-spread concepts. Multiagent Reinforcement Learning (MARL) has the potential to overcome scalability shortcomings by showing promising results in applications such as robot swarms (Hüttenrauch et al. 2017) and the aforementioned autonomous vehicles (Cao et al. 2012).

Despite the growing success of MARL, also due to the adoption of neural networks in RL (Tampuu et al. 2015), there is still little work on combining these multi-agent learning techniques with formal methods. This is the long-term challenge that we here start to address.

Our contribution in this chapter the extended Markov Games (EMGs), a non-markovian extension of the Markov Games (MGs), the mathematical model that is traditionally used in MARL. EMGs are a new general model, i.e, not focused solely in one kind of multi-agent game, that allows multiple learning agents to concurrently fulfill various non-Markovian specifications in multi-agent settings. To support our model with empirical evidence, we also extended two logic-based RL algorithms to multi-agents systems in order to show how various learning agents can fulfill different types of non-Markovian specifications expressed in *co-safe*- Linear-time Temporal Logic (*LTL*). Our results are promising and point to interesting future work we discuss in Section 4.5.

4.1 Preliminaries

In Section 2.4.2 we introduce Linear-Time Temporal Logic (LTL), an expressive and well-studied logic-based language that we will use to specify properties of MAS modelled as Markov games (MG).

We also present LTL_f and *co-safe* *LTL* (Kupferman & Vardi 2001) two variants of *LTL* that have been used in the recent works extending single-agent RL with temporal logic specifications (De Giacomo et al. 2018, Toro Icarte et al. 2018). Both are motivated by the fact that, in RL, agents are typically trained over finite episodes, but in different ways. The former is *LTL over finite traces* (LTL_f), which

is standard *LTL* interpreted over finite traces only, instead of infinite ones (De Giacomo & Vardi 2013). The latter is *co-safe LTL* (Kupferman & Vardi 2001), which – like *LTL* – is interpreted over infinite traces, but its syntax is restricted as follows:

$$\varphi ::= p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \text{U} \varphi_2 \quad (4.1)$$

The restricted syntax of *co-safe LTL* is meant to guarantee that formulas, if true, are true after a finite number of steps. We can show that *LTL_f* and *co-safe LTL* are actually equivalent, with the former restricting the semantics while the latter restricts the syntactic. The equivalence is stated in Corollary 1 below.

Corollary 1. *There are truth-preserving (polynomial) translations between *LTL_f* and *co-safe LTL*.*

Proof is given in Appendix D.

In this chapter we work with specifications expressed in *co-safe LTL*. The choice is based on working with finite episodes as we extend to multi-agent systems the algorithm LPOPL, which is already based on *co-safe LTL*. Nevertheless, as it can be inferred from Section 4.2, the approach presented here is not restricted to *LTL* and its variants, but applies to any formal language whose specifications can be transformed into a deterministic finite automata (DFA), that is, a finite-state machine that accepts or rejects finite strings of symbols (Rabin & Scott 1959).

Definition 4.1 (DFA). *A deterministic finite automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, q, \delta, F \rangle$, where Σ is the input alphabet, Q is the set of states with initial state $q_0 \in Q$, $\delta : \Sigma \times Q \rightarrow Q$ is the transition function, and $F \subseteq Q$ is the set of final states.*

For the mathematical framework presented in Section 4.2 to work, we will need to transform *co-safe LTL* specifications to a correspondent DFA. From Lacerda et al. (2015) we know that for any co-safe formula φ_k we can build a correspondent DFA $\mathcal{A}_\varphi = \langle 2^{\mathcal{A}^P}, Q_\varphi, q_{\varphi 0}, \delta_\varphi, F_\varphi \rangle$ that accepts exactly the finite traces that satisfy φ .

From here on, *co-safe LTL* specifications will be referred just as *LTL* specifications without abuse of notation since we proved that both *LTL_f* and *co-safe LTL* are actually equivalent.

LTL progression. As an agent progresses satisfying a multi-goal task, their tasks specifications may change. *LTL* formulas ϕ can be progressed (Bacchus & Kabanza 2000) along a sequence of truth assignments. *LTL* formulas can be progressed Bacchus & Kabanza (2000) along a sequence of truth assignments. In MARL, that means the formulas representing the specifications to be learned by the agents can be updated during an episode to reflect those requirements from the formulas that have been satisfied by the current history of states. Thus, progressed formulas include only those parts of the original problem that remain to be satisfied. For example, given the formula $\diamond(p \wedge \diamond q)$ (eventually p and then eventually q) can be progressed to $\diamond q$ once the agents reaches a state where p is true. We now introduce a formal definition of progression similarly to Bacchus & Kabanza (2000).

Definition 4.2. Given an *LTL* formula ϕ and a truth assignment σ_k over $\mathcal{AP}$, $\text{prog}(\sigma_k, \phi)$ is defined as follows:

- $\text{prog}(\sigma_k, p) = \text{true}$ if $p \in \sigma_k$, where $p \in \mathcal{AP}$
- $\text{prog}(\sigma_k, p) = \text{false}$ if $p \notin \sigma_k$, where $p \in \mathcal{AP}$
- $\text{prog}(\sigma_k, \neg\phi) = \neg\text{prog}(\sigma_k, \phi)$
- $\text{prog}(\sigma_k, \phi_1 \wedge \phi_2) = \text{prog}(\sigma_k, \phi_1) \wedge \text{prog}(\sigma_k, \phi_2)$
- $\text{prog}(\sigma_k, \bigcirc\phi) = \phi$
- $\text{prog}(\sigma_k, \phi_1 \cup \phi_2) = \text{prog}(\sigma_k, \phi_1) \vee (\text{prog}(\sigma_k, \phi_1) \wedge \phi_1 \cup \phi_2)$

Example 1. As running example throughout the paper, as well as for the experimental evaluation in Section 4.4, we consider a Minecraft-like grid world similar to the one introduced in Andreas et al. (2017) but extended with multiple learning agents. In this multi-agent scenario, the learning agents can interact with objects, extract raw materials from their environment and use them to manufacture new objects. This scenario also includes a set of features and events that are detectable by the agents: $\{\text{got_wood}, \text{used_toolshed}, \text{used_workbench}, \text{got_grass}, \text{used_factory}, \text{got_iron}, \text{used_bridge}, \text{used_axe}, \text{at_shelter}\}$ with obvious interpretations. This set becomes the set $\mathcal{AP}$ of atoms in our example. By using these atoms we can specify long-term goals in co-safe LTL, e.g., the specification of making shears can be expressed as:

$$\Phi_{shears} \triangleq \diamond (got_wood \wedge \diamond used_workbench) \wedge \diamond (got_iron \wedge \diamond used_workbench)$$

Notice that wood and iron can be collected in either order and that one workbench usage is enough to fulfill the task as long as it is done after collecting both items. We call this kind of specification an interleaving specification. We suppose that agents have to collaboratively fulfill a number of multi-task specifications, such as make shears by collecting wood and iron and using the workbench. We can model such a scenario as a Markov game, by defining S as a grid map representing the positions of the agents in N and the objects within the grid. Then, the actions in A_i available to each agent $i \in N$ will be the set $\{Up, Down, Left, Right, Wait\}$. Further, P will represent the probability of reaching a new state, i.e., a new distribution on the grid map, given the previous distribution and the joint action taken by the agents. The reward R_i will be the same for all agents and a positive reward would be given by the system when the relevant specification is satisfied. Note that, in order for this model to be Markovian, the satisfaction of the specifications must depend only on the last state and action taken. Finally, the discount factor γ will have a value between 0 or 1, depending on how important we want future rewards to be for the agents.

4.2 Non-Markovian Multi-Agent Specifications

Reinforcement Learning is designed to work under Markovian reward models. The main limitation of this kind of modeling is the *Markovian assumption*, i.e., rewards depend only on the last state visited and action taken. Thus we cannot adopt reward functions that capture conditional temporally extended properties on vanilla Markov games, such as *LTL* specifications. In this section we detail how to handle non-Markovian specifications with RL in general-purpose game systems.

4.2.1 Non-Markovian Reward Games

In this section we introduce the concept of non-Markovian reward game, which is similar to Markov games, but where the reward depends on the history of states visited and actions taken.

Definition 4.3 (NMRG). A non-Markovian reward game is a tuple $G = \langle S, N, A, P, \bar{R}, \gamma \rangle$ where

- S, N, A, P and γ are defined as for a Markov game in Definition 2.3;
- the reward $\bar{R}$ is a set of real-valued functions over state-action histories in $(S \times A_1 \times \dots \times A_N)^*$ (which we will refer as traces), i.e., for every $i \in N$, $\bar{R}_i : (S \times A_1 \times \dots \times A_N)^* \rightarrow \mathbb{R}$.

By Definition 4.3, NMRGs allow us to define rewards that depend on a full trace rather than just the last state and action taken. We are also working with *LTL* specifications so we embed them into the NMRG.

Definition 4.4. A NMRG with co-safe LTL specifications is a tuple $G = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ where:

- $\langle S, N, A, P, \bar{R}, \gamma \rangle$ is an NMRG;
- $\mathcal{L} : S \rightarrow 2^{\mathcal{AP}}$ is the labelling function;
- Φ is a set of co-safe LTL specifications.

Note that for the NMRG to be consistent, as it can be intuitively inferred, $\bar{R}_i \in \bar{R}$ should be correlated to the progression, satisfaction or violation of any specification $\varphi_k \in \Phi_i$. Where Φ_i is the set of specifications associated to agent i . The correlation between specification and reward function should be consistent with the following rule:

$$r_s \geq r_p > r_v \quad (4.2)$$

Where Φ_i is the set of specifications associated to agent i , r_s is the reward granted by $\bar{R}_i$ to the agent i when it satisfies any $\varphi_k \in \Phi_i$, r_p is the reward granted when progressing any of those φ_k and r_v is the one granted when violating them. By violation of a specification we mean that it cannot be fulfilled anymore within the current episode.

Henceforth, when we mention NMRGs, we will be referring to NMRGs with *LTL* specifications. Note that in a NMRG P and $\bar{R}$ are unknown to agents. Again, we cannot apply RL algorithms directly to NMRGs since this would likely lead to non-stationary policies that would not converge. Notice that RL is typically based on Markovian Models; thus, for most of the RL algorithms to work effectively, we need the reward functions to depend solely on the last state and action taken.

4.2.2 Extended Markov Games

In this work we want to learn general purpose non-Markovian *LTL* specifications in a multi-agent Markovian setting, so as to exploit MARL techniques. This section is devoted to introduce Extended Markov Games (EMG) as the mathematical model to achieve this objective.

Definition 4.5 (Problem statement). *Given a NMRG $\mathcal{G} = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ and $s_o \in S$ as the initial state, we want each agent to learn its own non-Markovian optimal policy $\bar{\pi}^*$ that for each state chooses the best action leading to satisfy the given co-safe LTL specification.*

Further, we introduce a *joint non-markovian policy* $\bar{\Pi}^*$ as the set of the agents' individual (non-markovian) policies $\{\bar{\pi}_1, \dots, \bar{\pi}_N\}$. We represent joint actions, i.e, sets of actions simultaneously taken by the agents, as $u \in \mathcal{U}$, where $u = \langle a_1, \dots, a_N \rangle$ and $\mathcal{U} = A_1 \times \dots \times A_N$.

Since we need to use a Markovian model to train our MARL agents we need a notion of equivalence between NMRGs and MGs in a similar fashion to what Bacchus et al. (1996) introduced for single-agent models.

Definition 4.6 (Equivalence). *A NMRG $\mathcal{G} = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ is equivalent to an extended MG $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma \rangle$ if there exist two functions $\tau : S' \rightarrow S$ and $\rho : S \rightarrow S'$ such that:*

1. *For all $s \in S$, $\tau(\rho(s)) = s$.*
2. *For all $s_1, s_2 \in S$ and $s'_1 \in S'$, if $P(s_1, u, s_2) > 0$ and $\tau(s'_1) = s_1$, there exists a unique $s'_2 \in S'$ such that $\tau(s'_2) = s_2$ and $P(s'_1, u, s'_2) = P(s_1, u, s_2)$.*
3. *For any feasible trajectory $\langle s_0, u_1, \dots, s_{l-1}, u_l, s_l \rangle$ of $\mathcal{G}$ and $\langle s'_0, u_1, \dots, s'_{l-1}, u_l, s'_l \rangle$ of $\mathcal{G}'$, such that $\tau(s'_t) = s_t$ and $\rho(s_0) = s'_0$, we have $\bar{R}(\langle s_0, u_1, \dots, s_{l-1}, u_l, s_l \rangle) = R'(\langle s'_0, u_1, \dots, s'_{l-1}, u_l, s'_l \rangle)$.*

where by feasible trajectory we refer to any trajectory that is consistent within the transitions in the game.

The crucial points in Definition 4.6 are clauses (2) and (3), which assert the equivalence of the two models (with respect to the initial states) in both their dynamics and reward structure. In particular,

clause (2) ensures that for any given trajectory in $\mathcal{G}$

$$s_0, u_1 \xrightarrow{P(s_0, u_1, s_1)} s_1, u_2 \cdots s_{l-1}, u_l \xrightarrow{P(s_{l-1}, u_l, s_l)} s_l$$

and for any extended state s'_0 with base state s_0 , i.e., $\tau(s'_0) = s_0$, there is a trajectory in $\mathcal{G}'$ of similar structure:

$$s'_0, u_1 \xrightarrow{P'(s_1, u_1, s_2)} s'_1, u_2 \cdots s'_{l-1}, u_l \xrightarrow{P'(s_{l-1}, u_l, s_l)} s_l$$

where $\tau(s'_t) = s_t$ for all t in the trajectory. In this case, trajectories $\langle s_0, u_1, \dots, s_l \rangle$ and $\langle s'_0, u_1, \dots, s'_l \rangle$ are called *weakly correspondent*. Clause (3) on the other hand, imposes requirements on the individual rewards assigned to each of the agents in the extended game $\mathcal{G}'$. If we have two weakly correspondent trajectories $\langle s_0, u_1, \dots, s_l \rangle$ and $\langle s'_0, u_1, \dots, s'_l \rangle$ such that $p(s_0) = s'_0$, we say that these trajectories are *strongly correspondent*. This also means that their relationship is one-to-one, that is, each trajectory in $\mathcal{G}$ has a unique strongly correspondent in $\mathcal{G}'$, and that a trajectory in $\mathcal{G}'$ has a unique strongly correspondent if its first state is an initial state. Thus, clause (3) requires that the functions in R' assign individual rewards to extended states in such a manner that strongly corresponding trajectories receive the same reward.

In a similar fashion to what Brafman et al. (2018) did extending MDPs, we introduce the following definition:

Definition 4.7 (Equivalent MG). *Given an NMRG $\mathcal{G} = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ with $\Phi = \{\phi_1, \dots, \phi_M\}$, let $\mathcal{A}_k = \langle 2^{\mathcal{A}^P}, Q_k, q_{k0}, \delta_k, F_k \rangle$ be the DFA corresponding to ϕ_k . We define the equivalent Markov game $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma \rangle$ such that*

- $S' = Q_1 \times \cdots \times Q_M \times S$ is the set of states;
- $\mathcal{U}, \Phi, \mathcal{L}$ and γ are defined as in the NMRG.
- $P' : S' \times \mathcal{U} \times S' \rightarrow [0, 1]$ is defined as follows:

$$P'(\vec{q}, s, u, \vec{q}', s') = \begin{cases} P(s, u, s') & \text{if for all } i : \delta_i(q_i, s') = q'_i \\ 0 & \text{otherwise} \end{cases}$$

- $R' : S' \times \mathcal{U} \times S' \rightarrow \mathbb{R}$ is defined as:

$$R'(\vec{q}, s, u, \vec{q}', s') = \sum_{i: q'_i \in F_i} r_i$$

Intuitively, by Definition 4.7 the extended state space in $\mathcal{G}'$ is a product of the states in the original NMRG and the automata for the various *LTL* formulas. Notice that, since we assume a fully observable environment, it is important that the new state-space perceived by every agent is composed by all automata. Given a joint action u , the s -component in the extended state progresses according to the original dynamics in the NMRG, while the transition function of the corresponding automaton marks the progress of the other components.

Theorem 1. *The NMRG $\mathcal{G} = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ is equivalent to the MG $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma \rangle$ defined above.*

Proof. Recalling that every $s' \in S'$ has the form $(q_1, \dots, q_M, s)$, we define $\delta(q_1, \dots, q_M, s) = s$ and $\sigma(s) = (q_1, \dots, q_M, s)$. Thus, $\delta(\sigma(s)) = s$. Condition 2 of Definition 4.6 can be easily verified by inspection. For condition 3, let's consider a trace $\eta = \langle s_0, u_1, \dots, s_{n-1}, u_n, s_n \rangle$. We use σ to obtain $s'_0 = \sigma(s_0)$ and given s_t we define s'_t (for $1 \leq t < n$) to be the unique state $(q_{1t}, \dots, q_{Mt}, s_t)$ such that $q_{jt} = \delta(q_{j,t-1}, u_t)$ for all $1 \leq j \leq M$. Thus, we now have a corresponding trace in $\mathcal{G}'$, i.e., $\eta' = \langle s'_0, u_1, s'_1, \dots, s'_{n-1}, u_n, s_n \rangle$, which is the only feasible trajectory in $\mathcal{G}'$ that satisfies Condition 3. The reward at η depends solely on whether or not each specification ϕ is satisfied by η . Nevertheless, by construction of the DFA $\mathcal{A}_k$ and the transition function P' , we have that $\eta \models \phi_k$ iff $s'_{n-1} = (q_1, \dots, q_M, s'_n)$ and $q_k \in F_k$ $\square$

Given a joint Markovian policy Π' for $\mathcal{G}'$, a policy $\bar{\Pi}$ on $\mathcal{G}$ that guarantees the same rewards can be easily found. To this end, consider a trace $\sigma = \langle s_0, u_1, s_1, \dots, s_{l-1}, u_l, s_l \rangle$ in $\mathcal{G}$ and let q_i be the state of A_i on the input σ . We define the (non-Markovian) joint policy $\bar{\Pi}$ equivalent to Π' as $\bar{\Pi}(\sigma) = \Pi'(q_1, \dots, q_M, s_l)$. Given this and Definition 4.6, we can define optimal policies for $\mathcal{G}$ by solving $\mathcal{G}'$ instead. Extending Bacchus et al. (1996) to a multi-agent setting we obtain:

Theorem 2. *Given a NMRG $\mathcal{G}$, let Π' be a set of optimal policies for an equivalent MG $\mathcal{G}'$. Then, the policy $\bar{\Pi}$ for $\mathcal{G}$ that is equivalent to Π' is optimal for $\mathcal{G}$.*

It can be then deduced that multi-agent RL techniques can be directly applied to $\mathcal{G}'$, so that we can obtain an optimal joint policy Π'^* for $\mathcal{G}'$. Therefore, an optimal joint policy for the NMRG $\mathcal{G}$ can be learnt from $\mathcal{G}'$.

Remark 1. *Note that none of these game structures is explicitly known by the learning agents, in the sense that the agents are never given the whole structure as input, but rather they just get to know what they can observe through their interactions. Thus, in practice the above transformations are never done explicitly. Instead, the agents learn by assuming that the underlying model is $\mathcal{G}'$.*

Since the state space of $\mathcal{G}'$ is the product of the state spaces of $\mathcal{G}$ and of the automata A_φ , the rewards in R' are Markovian, i.e., the state representation of the temporal formulas in Φ are compiled into the states of $\mathcal{G}'$. As in De Giacomo et al. (2018) for single agents, and given Cor. 1, we can state the following result in the multi-agent case:

Corollary 2. *RL for co-safe LTL rewards φ over an NMRG $\mathcal{G} = \langle S, N, A, P, \bar{R}, \mathcal{L}, \Phi, \gamma \rangle$ can be reduced to RL over the EMG $\mathcal{G}'$ in Definition 4.7.*

Remark 2. *We observe that Cor. 2 holds for specifications in Φ expressed in any temporal language as long as there is a way to compute a DFA for each of the specifications.*

This means that we can employ *LTL* as well as the variants we presented in Section 2.4.2, whose queries can be directly transform into DFAs, or even more expressive languages such as Linear Dynamic Logic over finite traces *LDL_f* (Brafman et al. 2018).

Example 2. *Consider the specification of making shears we made in our last example:*

$$\varphi_{\text{shears}} \triangleq \diamond (\text{got_wood} \wedge \diamond \text{used_workbench}) \wedge \diamond (\text{got_iron} \wedge \diamond \text{used_workbench})$$

If we want to reward our agents for making shears given the state space we first defined, we would need to check the state-action history of the agents when using the workbench to reward them only if they previously got wood and iron. Since rewards depend on the trace followed by the agents, our model would not be a Markov game anymore but an NMRG. In order to model our problem as an

Extended Markov Game, we transform the *LTL* specification of making shears into a *DFA* whose states would be given to the agents as an extension of the original observation that they perceive from the environment. The agents would then begin the episodes by perceiving an observation of the map extended by the initial state of the automaton, that represents the whole specification of making shears to be fulfilled. Once the agents have progressed the specification, which in this case means that they got iron or wood, the automaton will transit to a new state that represents the remainder of the specification to be fulfilled. The agents will notice a change in their perception of the environment because of this new state in the *DFA*. Hence, the agents will perceive the process as Markovian.

Remark 3. *Note that the dimensionality of the extended state space is not dependant on the number of specifications, as it extends Markov Games only with the representation of the current DFA state. Thus, following the methodology presented in our Section 4.4, we only need to extend the state space with a single feature. When working with multiple specifications, these can still be represented with a single DFA.*

4.3 Deep MARL with *co-safe LTL* Goals

To empirically support our theoretical results on EMGs, this section is devoted to introduce two decentralised extensions of single-agent DRL algorithms designed to work with *LTL* specifications. The first approach is based on extending with temporal logic specifications a popular baseline in MARL called I-DQL (Tampuu et al. 2015), while the second is a multi-agent extension of LPOPL (Toro Icarte et al. 2018) (we refer the reader to Chapter 3 for further detail). The extended algorithms described below are employed in the experiments presented in Section 4.4.

4.3.1 I-DQL with *co-safe LTL* Goals

The first algorithm is based on DQNs introduced in Section 2.3.1. Q-values for a given agent i can be defined in the multi-agent context (Foerster et al. 2017) by fixing the policy of all the other agents.

Let $Q_i^{\pi_i}$ be the Q-value for a given state s and action a for agent i , defined as follows:

$$Q_i^{\pi_i}(s_0, a_i \mid \Pi_{-i}) = \mathbb{E}_{\pi_i} \left[\sum_{t=0}^T \gamma^t r_t \mid s_0, a_i, \Pi_{-i} \right] \quad (4.3)$$

where Π_{-i} are the policies of all agents other than i , and T is the length of the trace.

Intuitively, Equation (4.3) states that the quality of an action-state pair, given a policy for an agent i , can be expressed as in the single-agent case given the policies of all other agents. This allows Independent Q-learning (Tan 1993) to train multiple agents in a decentralised fashion. Here we consider a deep learning variant of this algorithm (see, e.g., Tampuu et al. (2015)), where each agent is trained with an independent DQN. However, in our case, we adopt a decentralised version of an algorithm that uses *LTL* specifications and *LTL* progression instead of classical reward functions (see, e.g., Littman et al. (2017)). Hereafter we refer to this algorithm as I-DQL-L. Since we are working with sets of specifications, I-DQL-L also incorporates a curriculum learner that selects the specification that the agents will have to fulfill in the next episode. We selected a simple method that assumes the specifications in Φ follow some order $\varphi_0, \dots, \varphi_{M-1}$ where $|\Phi| = M$, and the curriculum learner just selects the next specification following the given order.

4.3.2 I-LPOPL

Our second algorithm for experimental evaluations is a multi-agent extension of LPOPL (Toro Icarte et al. 2018), which is designed specifically to take advantage of *co-safe LTL* specifications in order to boost agents learning performance. Similarly to what we described above for I-DQL-L, we combine the version of LPOPL that employs DQNs as function approximators with Independent Q-learning to obtain a centralised-decentralised algorithm we call Independent-LPOPL (I-LPOPL). Notice that, in MARL, centralised components refer to those that are shared among all the agents, while decentralised are those components that run locally for each agent. Similarly to the single agent algorithm, I-LPOPL is based on the same 4 core components, which we extend to multi-agent systems as detailed below:

- **A centralised task extractor.** We use the original task extractor from LPOPL in a centralised manner. This component receives as input the global set of specifications Φ , and returns an extended set Φ^+ of sub-specifications, or *tasks*, which contains each unique *LTL* task that the original set of specifications can be divided in, including the original set Φ of specifications. For instance, given $\Phi = \{\diamond(b \wedge \diamond c), \diamond(d \wedge \diamond a)\}$, after a first progression the specifications become $\diamond c$ and $\diamond a$. Thus, here the task extractor would generate the extended set $\Phi^+ = \{\diamond(b \wedge \diamond c), \diamond(d \wedge \diamond c), \diamond c, \diamond a\}$.
- **Decentralised LPOPL behavior policies.** In I-LPOPL each agent has its own set of networks, as each agent has a different DQN working as a Q-value function approximator for each extracted task. If ϕ is the specification to solve in the current episode and ϕ^+ the result of progressing ϕ through the history of the episode so far, then the behavior policy is ϵ -greedy on the I-DQL-L whose goal is to satisfy ϕ^+ , i.e, the behavior policy for each agent would be the one optimised to solve the current goal task observed by the agent in the extended state of the EMG.
- **Decentralised Q-value function updates.** In each transition, each agent updates its DQNs independently, that is, at each time step, the sequence $\langle s'_t, u_t, s'_{t+1}, r_{t+1} \rangle$ is stored in the replay buffer of each DQN, where r is given accordingly to the task each DQN is learning. However, in order to allow agents to develop different and potentially complementary individual policies, the update process for each agent is independent from the others'.
- **A centralised curriculum learner:** At the beginning of every iteration of I-LPOPL a global curriculum learning method is used to select the next specification to learn, in the same fashion as in I-DQL-L. To solve a formula Φ , we assume the specifications follow some sequential composition $\phi_0, \dots, \phi_{M-1}$ where $|\Phi| = M$, and the curriculum learner selects the next specification from the formula.

A training iteration of I-LPOPL proceeds as follows: the task extractor receives the set of specifications and creates the sets of tasks. Then, an independent DQN is initialised by each agent for each of these tasks. Once done, the curriculum learner selects the next specification to be solved by the agents in the environment. The agents start interacting with the environment, and once there are

enough interactions to start learning, the Q-value function update algorithm is called after each transition. When the curriculum learner triggers the end of the learning process for the current task, a new one is selected. This process is repeated through all specifications.

4.4 Experimental Evaluation

In this section, we provide empirical evidence that multiple agents can concurrently learn policies that satisfy multiple non-Markovian specifications by modelling the environment as an EMG.

4.4.1 Experimental Setup

This section is devoted to explain the testing environment, and the features used in our experiments. Our goal here is first to prove that multiple RL agents can be trained to solve multiple non-Markovian specifications, and second that agents can develop multi-agent policies, in this case collaborative policies, to solve the given specifications.

The Minecraft-like world. The environment where we test our algorithms is the Minecraft-like grid map described in Examples 1 and 2. We chose this environment as it is similarly used as testing set-up in De Giacomo et al. (2018), Toro Icarte et al. (2018), Andreas et al. (2017). Specifically, our environment and test setting is a multi-agent version of the one presented in Toro Icarte et al. (2018), where we introduce two agents that need to fulfill different sets of *co-safe LTL* specifications. Two agents cannot be in the same place at the same time, which means that agents can obstruct each other. We model the environment as a discounted reward problem including the relevant DFAs. That is, given the current specification ϕ to be solved and its associated DFA $\mathcal{A}_\phi$, a reward of -1 is given each time the DFA remains in the same state, 0 if the DFA transitions to a new intermediate state (i.e, the agents have progressed in the specification), and +1 if the DFA reaches a terminal state (i.e, the agents fulfilled the specification).

Features. The two algorithms consider the same features, actions, network architecture, and optimiser. The input features are contained in a vector that also registers the distance of every object to

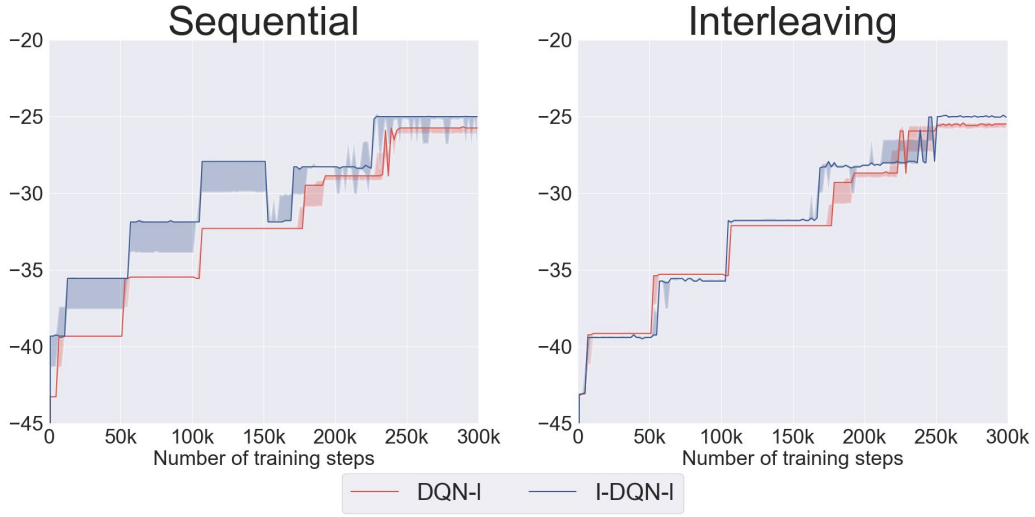


Figure 4.1: Rewards obtained by DQN-I and I-DQN-I greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 50k training steps per specification of the set.

the agent receiving the input, as in Andreas et al. (2017), Toro Icarte et al. (2018). The DQNs code is based on the implementation from OpenAI Baselines (Dhariwal et al. 2017). Specifically, we use a feedforward network with 2 hidden layers of 64 neurons using ReLu (Glorot et al. 2011) as activation function. The networks are trained using Adam optimisation (Kingma & Ba 2014) with a learning rate of 5×10^{-4} , sampling with a batch size of 32 transitions over a replay buffer with a sample size of 25,000 and updating the target networks every 100 steps. The discount factor for both DQN-I and I-DQN-I is 0.98, which works better with them since their networks are trained to solve a full specification; while the discount factor in both LPOPL and I-LPOPL is 0.9, which is a better choice for these algorithms, since each DQN is focused on solving just a sub-specification or short-term task.

4.4.2 Specification Sets

This section introduces and explains the different sets of specifications to be solved by the agents, also based on a multi-agent extension of Toro Icarte et al. (2018), Andreas et al. (2017):

- **Specifications as sequences of tasks.** The first set contains 10 specifications from Andreas et al. (2017) which are a simple sequence of properties to be satisfied by the two agents. For instance, the specification *make_shears* defined in Example 1, can be translated to a sequen-

tial *LTTL* specification as $\diamond(\text{got_wood} \wedge \diamond(\text{used_workbench} \wedge \diamond(\text{got_iron} \wedge \diamond \text{used_workbench})))$.

While a single agent can fulfill the whole sequence, an intuitively better policy would be one where one agent goes to get wood and the other waits at the workbench until the first agent collects the wood, then the agent closer to iron moves towards it and the other agent waits at the workbench.

- **Specifications as interleaving tasks.** This set, proposed in Toro Icarte et al. (2018), differs from the previous one as specifications are no longer a fixed sequence, but rather unnecessary ordering over parts of the specifications are removed. The *LTTL* specification that we used in Example 1 and 2 for making shears belongs to this set.

Note that, while we extend the experiments in Toro Icarte et al. (2018), our single-agent version of the algorithms do not match exactly the setting used in the reference. Specifically, we changed the discount factors, the learning rate, and the reward functions in order to optimise the learning times of the algorithms, a much needed step when doing decentralised multi-agent experiments due to scalability with the number of agents.

4.4.3 Experimental Results

In order to evaluate our approach, we ran the multi-agent algorithms introduced in Section 4.3 in a multi-agent version of one of the random maps from Toro Icarte et al. (2018), and tested them against the correspondent single-agent algorithm in an unaltered map. In both versions, the agents had a time-limit of 300 steps to solve the specification assigned to an episode. Both algorithms were trained for a given number of training steps per specification (see Figures 4.1 and 4.2). Once the limit was reached, the curriculum learner selected the next task to be learnt. Every 1k steps we measured the performance of the greedy policies developed by the algorithms on the whole set of specifications. Figures 4.1 and 4.2 shows the mean rewards obtained by these policies, and the shadowed areas are the 25th and 75th percentiles obtained across 3 independent runs per algorithm per set of tasks. Figure 4.1 compares the performance of I-DQN-l and DQN-l. These algorithms have a different policy network for each specification, where each network is updated only when the curriculum learner has selected

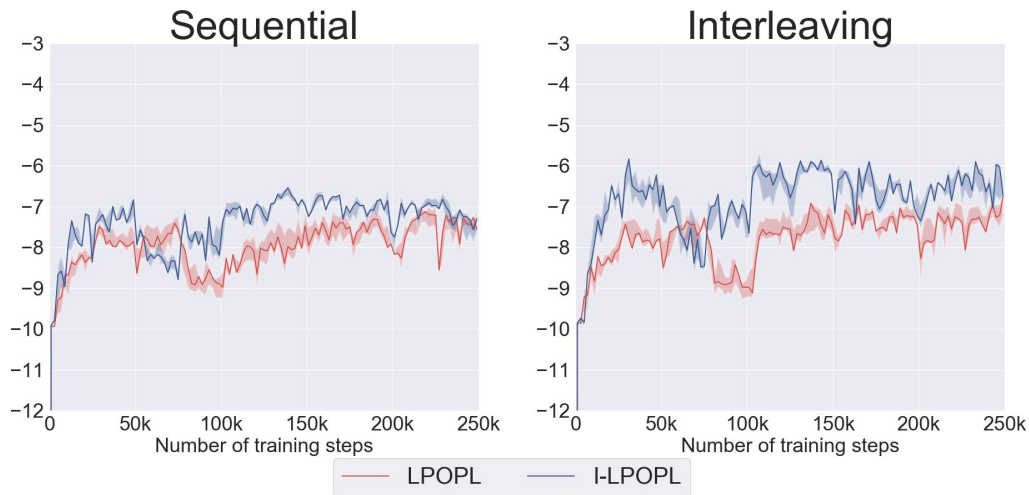


Figure 4.2: Rewards obtained by LPOPL and I-LPOPL greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 25k training steps per specification of the set.

the correspondent specification. This is why we observe steeped reward figures in the plot. I-DQN-I achieves higher rewards than DQN-I in the single-agent map thanks to the collaborative strategies that the agents develop in the multi-agent setting, i.e, if the specification requires to collect an item and then use a tool, typically one agent goes for the item while the other waits with the tool to use it as soon as the first agent has finished.

Figure 4.2 show the performance of I-LPOPL and LPOPL. Again, the higher rewards obtained by I-LPOPL over LPOPL are due to agents collaborating in the multi-agent setting. These algorithms do not show steeped reward figures because at each training step every network is updated accordingly to its assigned specification. That is, LPOPL variants can improve its performance on all the specifications at the same time. We note, however, that this policy update system is also causing some drops in the performance of the multi-agent algorithm, for instance after the 50k step. This because strategies that are good for the online policy (the want learning about the current specification) might learn something that is not good for the other policies. For instance, in our experiments it happened that each agent learned that they should just navigate towards the closest object. However, the offline policies do not take into account that for some specifications one agent can obstruct the other, making the behavior less effective. Thus, the agent will in some cases forget the collaborative policies and learn single-agent ones offline, causing drops in performance as those in Figure 4.2.

Finally, we highlight that while I-LPOPL learns faster than I-DQN-I in terms of training steps, I-

Table 4.1: Computing times of the different algorithms and variants studied in this chapter.

Algorithm	Sequential	Interleaving
DQN-I	1.20	1.30
I-DQN-I	1.57	2.15
LPOPL	5.13	6.5
I-LPOPL	11.32	14.28

LPOPL demands higher computing resources and needs longer time to train. This is because in I-DQN-I we have a DQN per agent per specification, while in I-LPOPL-I we have a DQN per agent per sub-specification. In our experiments, we had 10 specifications in both set of tasks, 27 sub-specifications in the sequential set and 34 in the interleaving set. Table 4.1 below shows the computing time for each algorithm in the experiments. The experiments were run on a laptop with a GeForce RTX 2080 as GPU, 32 GB of RAM and a i7 7700 HQ as CPU, computing the three independent runs in parallel.

4.5 Discussion

In this chapter we introduced Extended Markov Games, a mathematical model for multi-agent reinforcement learning to learn policies that satisfy multiple (non-Markovian) *LTL* specifications in multi-agent systems. Our formal definitions actually imply that any temporal logic can be used to express the specifications as long as they can be converted to DFAs. Our experimental results in cooperative games provides empirical evidence that the proposed framework allows RL agents to learn and develop multi-agent strategies to satisfy different sets of specifications. These results facilitated the foundations for MAS-specific RL algorithms Hammond et al. (2021) and more effective logic based structures for MARL (Neary et al. 2021). More recently, Ardon et al. (2023) also extended reward machines (Icarte et al. 2018), a more expressive alternative to *co-safe LTL* and DFA, to multi-agent systems. One limitation of the work in this chapter is that we did not address environments with partial observability. This is the central topic of Chapter 5.

Chapter 5

Non-Markovian Task Execution Under Imperfect Information

Sometimes, what you ignore about the past has an effect on your future. Chapter based on Belardinelli et al. (2022).

The problem of solving non-Markovian problems through Markovian techniques was first tackled in Bacchus et al. (1996). Therein, the authors solved an MDP with non-Markovian rewards (NMRDP) by generating an equivalent MDP so that an optimal policy in the latter is also optimal for the former. Building on this, Brafman et al. (2018), Giacomo et al. (2019), and Chapter 4 in this thesis, extended the scope and applications of this method. Still, those line of works focus on fully observable environments, where the agent has perfect knowledge of the current state of the system. Assuming complete (state) knowledge of the environment is often unrealistic or computationally costly (Badia et al. 2020, Samvelyan et al. 2019), and while there exists successful empirical works applying the formalism from Bacchus et al. (1996) or similar extensions in scenarios of imperfect knowledge (see Chapters 6, 7, or also Icarte et al. (2019)), there is no theoretical analysis on which conditions these extensions should fulfill to guarantee solving the original non-Markovian model under imperfect state knowledge (also called partial observability or imperfect information).

Contribution. In this chapter we present a novel theoretical framework to solve a partially-observable NMRDP (PONMRDP) by building an equivalent partially-observable MDP (POMDP), while providing formal guarantees that any policy that is optimal in the latter has a corresponding policy in the former that is also optimal. We also prove that naive extensions of existing methods for fully-observable models can induce optimal policies that are not applicable in the original non-Markovian problem.

5.1 Partially Observable Models

In this section we recall standard notions on partially observable (p.o.) Markov decision processes (POMDPS) that were introduced in Chapter 2 and their non-Markovian version. Furthermore, for both models we present their belief-state counterparts, policies, and policy values. Given an element U in an MDP, $\bar{U}$ denotes its non-Markovian version, and U_b denotes its belief version. Intuitively, a belief version denotes a weighted average of the possible states the agent could be in a given timestep given a history of observations. Given a tuple $\vec{w}$, we denote its length as $|\vec{w}|$, and its i -th element as $\vec{w}_{i-1}$. Then, $last(\vec{w}) = v_{|\vec{w}|-1}$ is the last element in $\vec{w}$. For $i \leq |\vec{w}| - 1$, let $\vec{w}_{\geq i}$ be the suffix $w_i, \dots, w_{|\vec{w}|-1}$ of $\vec{w}$ starting at w_i and $\vec{w}_{\leq i}$ its prefix $w_0, \dots, w_i$. Moreover, we denote with $\vec{w} \cdot \vec{w}'$ the concatenation of tuples $\vec{w}$ and $\vec{w}'$. Finally, given a set V , we denote with V^+ the set of all non-empty sequences on V .

Definition 5.1 (PONMRDP). *A p.o. non-Markovian reward decision process is a tuple $\bar{\mathcal{M}} = \langle S, A, P, \bar{R}, Z, O, \gamma \rangle$, where:*

- S, A, P, Z, O and γ are defined as in Definition 2.2.
- A is the finite set of actions.
- $T : S \times A \times S \rightarrow [0, 1]$ is the transition probability function that returns the probability $T(s' | s, a)$ of transitioning to the successor state s' , given the previous state s and action a taken by the agent.
- $\bar{R}$ is the non-Markovian reward function defined as $\bar{R} : (S \cdot A)^+ \cdot S \rightarrow \mathbb{R}$.
- Z is the set of observations.

- $O : S \times A \times S \rightarrow Z$ is the observation function that given the current state s and action a , draws an observation z , based on the successor state s' .
- $\gamma \in (0, 1]$ is the discount factor.

Note that, we consider a deterministic observation function for simplicity of presentation. However, the environment is still stochastic due to the transition function. We adopt this modelling since it is the standard approach for RL algorithms working in p.o. scenarios (Rashid et al. 2018, Icarte et al. 2019, Vinyals et al. 2019, Zhao et al. 2021)

A *trajectory* $s_0, a_1, \dots, s_n \in (S \cdot A)^+ \cdot S$, denoted as $\vec{s}$, is a finite (non-empty) sequence of states and actions, ending in a state, where for each $1 \leq i \leq n$, $P(s_i | s_{i-1}, a_i) > 0$. By Definition 5.1, a p.o. Markov decision process (POMDP) is a PONMRDP where the reward function only depends on the last transition. That is, the reward function is a function $R : S \times A \times S \rightarrow \mathbb{R}$.

Because the agent cannot directly observe the state of the environment, they have to make decisions under uncertainty about their actual state. Then the agent updates her beliefs by interacting with the environment and receiving observations. A *belief state* b is a probability distribution over the set of states, that is, $b : S \rightarrow [0, 1]$ such that $\sum_{s \in S} b(s) = 1$.

Definition 5.2 (Belief update). *Given a belief state b , action a , and observation z , we define an update operator ρ such that $b' = \rho(b, a, z)$ iff for every state $s' \in S$,*

$$b'(s') = \begin{cases} \eta \sum_{s \in S} P(s' | s, a) b(s) & \text{if } O(s, a, s') = z; \\ 0 & \text{otherwise.} \end{cases} \quad (5.1)$$

where $b(s)$ denotes the probability that the environment is in state s ; $\eta = 1 / \Pr(z | b, a)$ is the normalisation factor for $\Pr(z | b, a) = \sum_{\{s' \in S | O(s, a, s') = z\}} \sum_{s \in S} P(s' | s, a) b(s)$.

Now, we have all the ingredients to define a belief non-Markovian reward decision process.

Definition 5.3 (BNMRDP). *Given a PONMRDP $\overline{\mathcal{M}}$, the corresponding belief NMRDP is a tuple $\overline{\mathcal{M}}_b = \langle B, A, T_b, \overline{R}_b, \gamma \rangle$ where:*

- A and γ are defined as in Definition 5.1.
- B is the set of belief states over the states of $\overline{\mathcal{M}}$.
- $P_b : B \times A \times B \rightarrow [0, 1]$ is the belief state transition function defined as:

$$P_b(b, a, b') = \sum_{z \in Z} Pr(b' | b, a, z) Pr(z | b, a)$$

$$\text{where } Pr(b' | b, a, z) = \begin{cases} 1 & \text{if } b' = \rho(b, a, z); \\ 0 & \text{otherwise.} \end{cases}$$

- the reward function $\overline{R}_b : (B \cdot A)^+ \cdot B \rightarrow \mathbb{R}$ is defined as:

$$\overline{R}_b(b_0, a_1, \dots, b_n) = \sum_{s_0, \dots, s_n \in S} b_1(s_0) \dots b_n(s_n) \overline{R}(s_0, a_1, \dots, s_n)$$

By Definition 5.3, given a POMDP $\mathcal{M}$, the corresponding belief MDP is a tuple $\mathcal{M}_b = \langle B, A, T_b, R_b, \gamma \rangle$, where The reward function $R_b : B \times A \times B \rightarrow \mathbb{R}$ is defined as

$$R_b(b, a, b') = \sum_{s, s' \in S} b(s) b'(s') R(s, a, s') \quad (5.2)$$

Remark 4. In what follows we assume w.l.o.g. an initial distribution b_0 . Such belief state b_0 can be generated by assuming an initial state s_0 and suitable auxiliary transitions over the states in b_0 .

5.1.1 Navigating PONMRDPs

We now define policies, policy values, and optimal policies in a similar fashion to Sutton & Barto (2018) for all frameworks described above.

Definition 5.4 (Non-Markovian Policy). A non-Markovian policy $\overline{\pi} : (S \cdot A)^+ \cdot S \rightarrow A$ is a function from trajectories to actions. The value $v^{\overline{\pi}}(\vec{s})$ of a trajectory $\vec{s}$ following a non-Markovian policy $\overline{\pi}$ is

defined as:

$$v^{\bar{\pi}}(\vec{s}) = \sum_{s' \in S} P(s' | \text{last}(\vec{s}), \bar{\pi}(\vec{s})) \left[\bar{R}(\vec{s}, \bar{\pi}(\vec{s}), s') + \gamma v^{\bar{\pi}}(\vec{s} \cdot s') \right]$$

An optimal non-Markovian policy $\bar{\pi}^*$ is one that maximises the expected value for any given trajectory $\vec{s} \in (S \cdot A)^+ \cdot S$, that is, for all $\vec{s}$, $v^{\bar{\pi}^*}(\vec{s}) \doteq \max_{\bar{\pi}} v^{\bar{\pi}}(\vec{s})$.

By Definition 5.4, a Markovian policy $\pi : S \rightarrow A$ is a non-Markovian policy that only depends on the last visited state. Then, the value $v^{\pi}(s)$ of a Markovian policy π at state s is:

$$v^{\pi}(s) = \sum_{s' \in S} P(s' | s, \pi(s)) \left[R(s, \pi(s), s') + \gamma v^{\pi}(s') \right]$$

Finally, an optimal Markovian policy π^* is such that for all $s \in S$, $v^{\pi^*}(s) \doteq \max_{\pi} v^{\pi}(s)$.

Definition 5.5 (non-Markovian Belief Policy). A non-Markovian belief policy $\bar{\pi}_b : (B \cdot A)^+ \cdot B \rightarrow A$ is a function from belief trajectories to actions. The value $v^{\bar{\pi}_b}(\vec{b})$ of a non-Markovian belief policy $\bar{\pi}_b$ for a trajectory $\vec{b}$ of belief states is defined as:

$$v^{\bar{\pi}_b}(\vec{b}) = \sum_{b' \in B} P_b(b' | \text{last}(\vec{b}), \bar{\pi}_b(\vec{b})) \left[\bar{R}_b(\vec{b}, \bar{\pi}_b(\vec{b}), b') + \gamma v^{\bar{\pi}_b}(\vec{b} \cdot b') \right]$$

An optimal non-Markovian belief policy in a p.o. model is one that achieves the maximum value for any trajectory of belief state $\vec{b} \in (B \cdot A)^+ \cdot B$: $v^{\pi^*}(\vec{b}) \doteq \max_{\bar{\pi}_b} v^{\bar{\pi}_b}(\vec{b})$.

By Definition 5.5, a Markovian belief policy $\pi_b : B \rightarrow A$ is a non-Markovian belief policy that only depends on the last belief state. Then, the value $v^{\pi_b}(b)$ of a Markovian belief policy π_b at belief state b is given as:

$$v^{\pi_b}(b) = \sum_{b' \in B} P_b(b' | b, \pi_b(b)) \left[R_b(b, \pi_b(b), b') + \gamma v^{\pi_b}(b') \right]$$

Finally, an optimal Markovian belief policy is such that for all $b \in B$, $v^{\pi_b^*}(b) \doteq \max_{\pi_b} v^{\pi_b}(b)$.

Notice that, in a PONMRDP, i.e., a model where the rewards depend on trajectories of states and actions, optimal policies might also be Markovian as in the case of the counterexample in Sec 5.3.

In following sections, when not explicitly stated, we will assume an element to be Markovian, e.g. we will refer to $\bar{\pi}$ as a non-Markovian policy and π as a policy.

5.2 Extended POMDPs

In this section, we describe a state-space extension method to generate a correspondence between POMDPs and PONMRDPs, which we call extended POMDPs (EPOMDPs). In particular, we extend the method originally proposed in Bacchus et al. (1996) to partial observability.

First of all, we show how to generate a belief trajectory from a state trajectory.

Definition 5.6 (Belief trajectory). *Given a model $\mathcal{M}$, let $s_0 \xrightarrow{a_1, p_1} s_1 \cdots s_{n-1} \xrightarrow{a_n, p_n} s_n$ be a trajectory, where $p_i = P(s_i \mid s_{i-1}, a_i)$, for all $1 \leq i \leq n$. We define the corresponding belief trajectory $b_0 \xrightarrow{a_1, p'_1} b_1 \cdots b_{n-1} \xrightarrow{a_n, p'_n} b_n$, where $p'_i = T_b(b_i \mid b_{i-1}, a_i)$ and $b_i = \rho(b_{i-1}, a_i, z_i)$ for all $0 < i \leq n$ ¹.*

Note that, in Definition 5.6, we make use of observations. In particular, the observation z_i is generated by the observation function and the trajectory, i.e., $z_i = O(s_{i-1}, a_i, s_i)$, and it is used in the belief update function to generate the next belief state.

Now, we can define our notion of expansion.

Definition 5.7 (EPOMDP). *A POMDP $\mathcal{M} = \langle ES, A, T_{ES}, R, EZ, O_{ES}, \gamma \rangle$ is an expansion of a PONMRDP $\overline{\mathcal{M}} = \langle S, A, T, \bar{R}, Z, O, \gamma \rangle$, if there exist functions $\tau : ES \rightarrow S$, $\sigma : S \rightarrow ES$, $\tau' : EZ \rightarrow Z$, and $\sigma' : Z \rightarrow EZ$ such that:*

1. *For all $s \in S$, $\tau(\sigma(s)) = s$.*
2. *For all $s_1, s_2 \in S$ and $es_1 \in ES$, if $T(s_2 \mid s_1, a) > 0$ and $\tau(es_1) = s_1$, then there exists a unique $es_2 \in ES$ such that $\tau(es_2) = s_2$ and $T_{ES}(es_2 \mid es_1, a) = T(s_2 \mid s_1, a)$.*
3. *For all $z \in Z$, $\tau'(\sigma'(z)) = z$.*

¹As detailed in Remark 4, we define an auxiliary initial state s_0 with suitable auxiliary transitions. Consequently b_0 is assumed to be the distribution assigning 1 to s_0 and 0 to all other states.

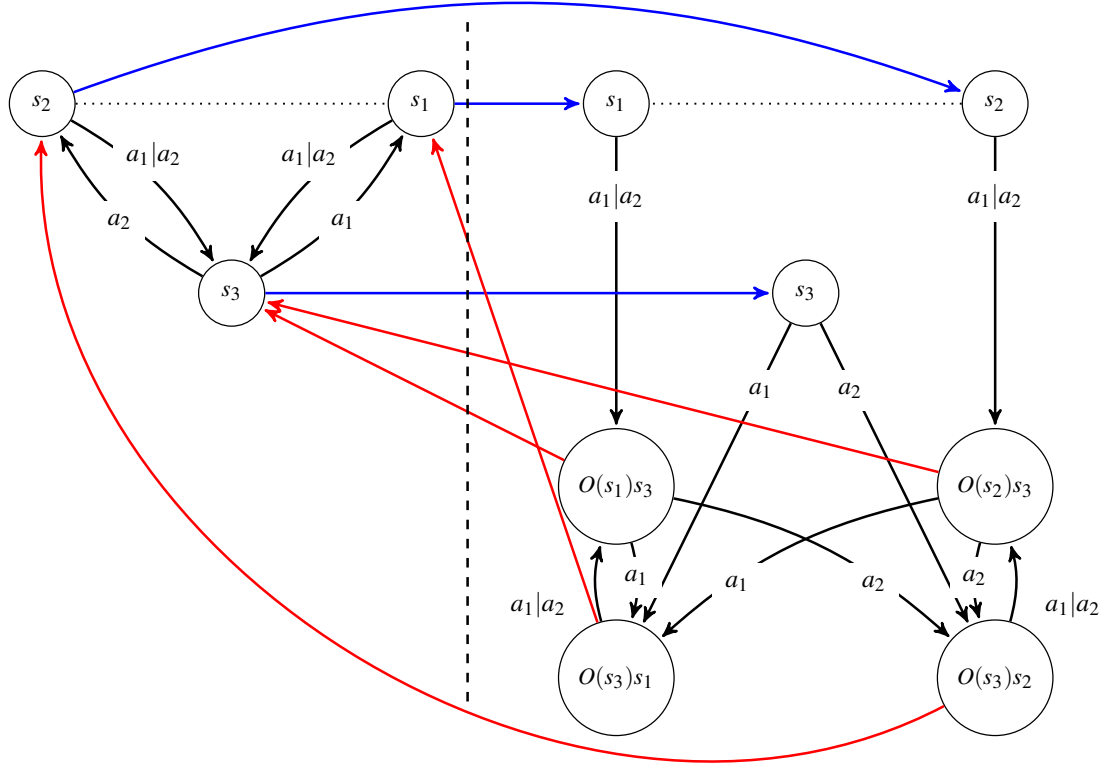


Figure 5.1: The models for the PONMRDP (left side) and an extended POMDP (right side) following Def. 5.7. Red lines denote some $\tau(es)$ transformations while blue lines we illustrate the cases in which $\sigma(s)$ is only the current state.

4. For all $z \in Z$, $s, s' \in S$, and $es, es' \in ES$, if $O(s, a, s') = z$, $\tau(es) = s$, and $\tau(es') = s'$, then there exist a unique $ez \in EZ$ such that $\tau'(ez) = z$ and $O_{ES}(es, a, es') = ez$.
5. For any trajectory $s_0, \dots, s_n$ in $\overline{\mathcal{M}}$ with corresponding belief trajectory $b_0, \dots, b_n$ as per Definition 5.6, the trajectory $es_0, \dots, es_n$ in $\mathcal{M}$ that satisfies $\tau(es_i) = s_i$ for each $0 \leq i \leq n$ and $\sigma(s_0) = es_0$, generates a belief trajectory $eb_0, \dots, eb_n$ such that $\bar{R}_b(b_0, \dots, b_{n-1}, a, b_n) = R_b(eb_{n-1}, a, eb_n)$.

Intuitively the extended state es and observation ez such that $\tau(es) = s$ and $\tau'(ez) = z$, can be thought of as labelled by $s \circ l$ and $z \circ l'$, where $s \in S$ is the base state (i.e., a state of $\overline{\mathcal{M}}$), $z \in Z$ is the base observation (i.e., an observation from $\overline{\mathcal{M}}$), and l and l' are labels that distinguish es and ez from other extensions of the same base elements. Figure 5.1 illustrates a POMDP and an extended POMDP that follow Definition 5.7.

The most important items in Definition 5.7 are points 2, 4 and 5. Points 2 and 4 ensure that both $\mathcal{M}$

and $\overline{\mathcal{M}}$ are equivalent regarding their respective base elements in both state and observation dynamics. Point 5 asserts equivalence in reward structure from the agent's perspective. Note that, differently from the expansions in fully observable models (Bacchus et al. 1996, Brafman et al. 2018), it is not enough that equivalent trajectories have the same rewards. Here, we require that they induce equivalent *beliefs* rewards (for details see Sec. 5.3).

An optimal agent, i.e., an agent that always follows an optimal policy, acting in $\overline{\mathcal{M}}$ would construct a BNMRDP, where it follows the policy that maximises the value of each belief state according to $\overline{R}_b$. Similarly, an optimal agent acting in $\mathcal{M}$ would construct an extended BMDP maximising the discounted expectation of R_b .

Since belief states are generated exclusively from transition probabilities and observations given states and actions, clauses 2 and 4 induce a similar behavior between equivalent trajectories of belief states to what is presented in Bacchus et al. (1996) for trajectories of states.

Lemma 1. *Let the POMDP $\mathcal{M}$ be an extension of the PONMRDP $\overline{\mathcal{M}}$. Given a belief trajectory $\vec{b}$ in $\overline{\mathcal{M}}$: $b_0 \xrightarrow{a_1, p_{b1}} b_1 \cdots b_{n-1} \xrightarrow{a_n, p_{bn}} b_n$, there is a trajectory $\vec{eb}$ in $\mathcal{M}$ defined as: $eb_0 \xrightarrow{a_1, p_{b1}} eb_1 \cdots eb_{n-1} \xrightarrow{a_n, p_{bn}} eb_n$ where $eb_i(es_i) = b(\tau(es_i))$ for all $0 \leq i \leq n$.*

We say that $\vec{b}$ and $\vec{eb}$ are weakly corresponding belief trajectories.

Proof. From clause 2 immediately follows that for any trajectory $\vec{s}$ in $\overline{\mathcal{M}}$

$$\vec{s} = s_0 \xrightarrow{a_1, p_1} s_1 \cdots s_n \xrightarrow{a_n, p_n} s_n$$

there is a trajectory $\vec{es}$ in $\mathcal{M}$ of similar structure

$$\vec{es} = es_0 \xrightarrow{a_1, p_{e1}} s_1 \cdots s_n \xrightarrow{a_n, p_{en}} s_n$$

where $p_{ei} = p_i$ and $\tau(es_i) = s_i$, for all $0 \leq i \leq n$. In this case we say that $\vec{s}$ and $\vec{es}$ are *weakly corresponding* trajectories. Symmetrically, clause 4 assures that these weakly correspondent real states trajectories will generate as well weakly correspondent observation trajectories. As a consequence, it follows immediately that these trajectories will generate weakly correspondent belief trajectories. $\square$

Given Lemma 1 we can define strong correspondence.

Definition 5.8. Let $\vec{b}$ and $\vec{eb}$ be weakly corresponding trajectories with initial belief states b_0 and eb_0 , respectively. We say that $\vec{b}$ and $\vec{eb}$ are strongly correspondent when eb_0 is an initial belief state.

This means that when the transformation induced by σ , i.e., $eb_i(\sigma(s_0)) = b_i$ is a trajectory containing only the current belief state, i.e, the first state of the trajectory in the non-Markovian model is the first state in the environment, we have strongly correspondent belief trajectories. Note that clause 5 requires that strongly correspondent belief trajectories have the same rewards.

Now we can introduce corresponding policies.

Definition 5.9 (Corresponding Policy). Let π_b be a belief policy for expansion $\mathcal{M}$. The corresponding belief policy $\bar{\pi}_b$ for the PONMRDP $\bar{\mathcal{M}}$ is defined as $\bar{\pi}_b(\vec{b}) = \pi'_b(\text{last}(\vec{eb}))$, where $\vec{eb}$ is the strongly corresponding trajectory for $\vec{b}$.

Given the expanded MDP from Definition 5.7 and corresponding policies as in Definition 5.9, we can now present the following result.

Proposition 1. For every policy π_b in expansion $\mathcal{M}$, corresponding policy $\bar{\pi}_b$ in PONMRDP $\bar{\mathcal{M}}$, we have that $v^{\bar{\pi}_b}(\vec{b}) = v^{\pi_b}(\text{last}(\vec{eb}))$, where $\vec{eb}$ and $\vec{b}$ are strongly corresponding trajectories.

Proof. This is evident since corresponding policies will generate the same actions for any corresponding belief trajectory and predict the same expected discounted return given the equivalent transition and observation dynamics from clauses 1-4 in Definition 5.7 and that clause 5 imposes the same belief rewards. □

Consequently we can find optimal policies for the PONRMDP by working on the POMDP instead.

Corollary 3. Let π_b an optimal policy for expansion $\mathcal{M}$. Then the corresponding policy $\bar{\pi}_b$ is optimal for the PONMRDP $\bar{\mathcal{M}}$.

Thus, given an optimal policy in the EPOMDP one can easily obtain an optimal solution for the original PONMRDP. As in previous fully-observable approaches, there is no need to generate $\bar{\pi}_b$ explicitly and, instead, the agent can run with π_b while assuming that the underlying model is $\mathcal{M}$.

5.3 A Counterexample to a Naive Extension

As anticipated in the introduction, we illustrate now the relevance of item 5 in Definition 5.7. In particular, equivalences under perfect knowledge require only that strongly correspondent trajectories of states have the same rewards (Bacchus et al. 1996, Brafman et al. 2018). Here we show that a naive extension of the definition of expansion in Bacchus et al. (1996), i.e., one that requires equivalent dynamics on states and observations, and enforces equivalence between rewards on state trajectories only, may induce policies that are unfeasible in the non-Markovian model. Formally, consider a variant of Definition 5.7, where item 5 only is replaced as follows:

5'. For every trajectory $s_0, \dots, s_n$ in $\overline{\mathcal{M}}$ and $es_0, \dots, es_n$ in $\mathcal{M}$ such that $\tau(es_i) = s_i$ for each $0 \leq i \leq n$ and $\sigma(s_0) = es_0$, we have $\overline{R}(s_0, \dots, s_{n-1}, a_{n-1}, s_n) = R(es_{n-1}, a_{n-1}, es_n)$.

We also introduce the notion of feasible policies:

Definition 5.10 (Feasible Policy). *Given a PONMRDP $\overline{\mathcal{M}}$ with an extended POMDP $\mathcal{M}$. We say that a policy π in $\mathcal{M}$ is not feasible in $\overline{\mathcal{M}}$ if there exist a pair of states s in $\overline{\mathcal{M}}$ and es in $\mathcal{M}$, where $\vec{s}$ is a trajectory in $\overline{\mathcal{M}}$ ending in s , $\tau(es) = s$ and $v^\pi(es) \neq v^{\overline{\pi}}(\vec{s})$ for every policy $\overline{\pi}$ in $\overline{\mathcal{M}}$.*

Now, we can prove the following theorem.

Theorem 5.1. *There exist a PONMRDP $\overline{\mathcal{M}}$ with expansion $\mathcal{M}$ given as in Definition 5.7 with item 5 replaced by item 5', where the optimal policy in $\mathcal{M}$ is not feasible in $\overline{\mathcal{M}}$.*

Proof. Consider the PONMRDP $\overline{\mathcal{M}} = \langle S, A, T, \overline{R}, Z, O, \gamma \rangle$ depicted in Fig. 5.2, such that:

1. $S = \{s_1, s_2, s_3\}$;
2. $A = \{a_1, a_2\}$;
3. $T^{a_1} = \{[0, 0, 1], [0, 0, 1], [1, 0, 0]\}$ and $T^{a_2} = \{[0, 0, 1], [0, 0, 1], [0, 1, 0]\}$, where $T^{a_1}[1] = T^{a_1}(s_1)[s_1, s_2, s_3]$, $T^{a_1}[2] = T^{a_1}(s_2)[s_1, s_2, s_3], \dots$;

4. the reward function $\bar{R}$ is given as

$$\bar{R}(\vec{w}) = \begin{cases} 1 & \text{if } \vec{w} = s_0, s_3, s_1, \text{ and } w_0 \neq w_1; \\ 1 & \text{if } \vec{w} = s_0, s_3, s_2, \text{ and } w_0 \neq w_2; \\ 0 & \text{otherwise.} \end{cases}$$

5. the observation function O is such that $O(s_1) = O(s_2) = z_1$ and $O(s_3) = z_2$ for $Z = \{z_1, z_2\}$ (we assume that observations are independent from the action taken);

6. $\gamma = 1$.

Finally, we assume an initial distribution $b_0 = [0.3, 0.7, 0]$. Note that, we consider time horizons of 3 time steps only.

Since both initial states s_1 and s_2 are indistinguishable (i.e., they return the same observation), but there is a higher chance of starting in s_2 , an optimal policy in $\bar{\mathcal{M}}$ is taking action a_1 for every state.

$$\pi^*(b_i) = [p(a_1) = 0, p(a_2) = 1] \text{ for any } b_i \quad (5.3)$$

Now consider the expansion $\mathcal{M}$, where states and observations are extended with a labelling that tells the agent the previous state visited. Formally, $\mathcal{M} = \langle ES, A, T_{ES}, R, EZ, O_{ES}, \gamma \rangle$, where:

- A and γ are the same as in $\bar{\mathcal{M}}$;
- $ES = \{s_1, s_2, s_1s_3, s_2s_3, s_3s_1, s_3s_2\}$;
- $EZ = \{ez_1, ez_2, ez_3, ez_4, ez_5\}$;
- $O_{ES}(\emptyset, \emptyset, s_1) = O_{ES}(\emptyset, \emptyset, s_2) = ez_1$ and for all $a \in A$, $es \in ES$, $O_{ES}(es, a, s_3s_1) = ez_2$, $O_{ES}(es, a, s_2s_1) = ez_3$, $O_{ES}(es, a, s_3s_2) = ez_4$, $O_{ES}(es, a, s_3s_2) = ez_5$.
- The transition function is depicted as in Figure 5.2.

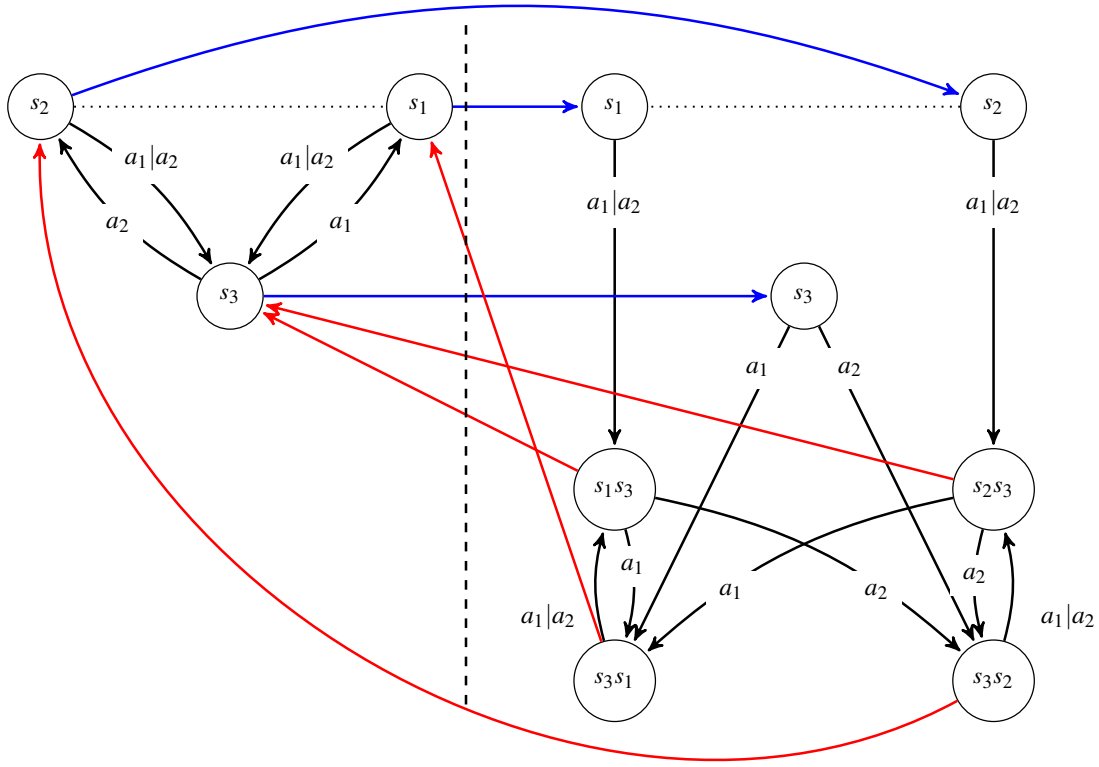


Figure 5.2: Counterexample to naive extension. The models for the PONMRDP (left side) and its extended POMDP (right side), satisfying item 5' instead of the proposed item 5 from Definition 7. Red lines denote some $\tau(es)$ transformations while blue lines illustrate the cases in which $\sigma(s)$ is only the current state.

- The reward function is defined as:

$$R(es_i, a_i, es_{i+1}) = \begin{cases} 1 & \text{if } es_i = s_1s_3, a_i = a_2, es_{i+1} = s_3s_2; \\ 1 & \text{if } es_i = s_2s_3, a_i = a_1, es_{i+1} = s_3s_1; \\ 0 & \text{otherwise.} \end{cases}$$

Now, functions $\tau : ES \rightarrow S$, $\sigma : S \rightarrow ES$, $\tau' : EZ \rightarrow Z$, and $\sigma' : Z \rightarrow EZ$ are intuitively defined as follows: τ and τ' add the information of the previous state visited to construct the extended state or observation respectively, while σ and σ' remove that information. The POMDP $\mathcal{M}$ satisfies the requirements in Definition 5.7 with condition 5' to be an expansion of $\overline{\mathcal{M}}$. Figure 5.2 illustrates $\mathcal{M}$ and $\overline{\mathcal{M}}$.

We now show that the optimal policies in this model is not feasible in the original. In the EPOMDP

$\mathcal{M}$ an optimal policy is one that:

$$\pi_b^*(eb_i) = \begin{cases} a_2 & \text{if } ez_i = ez_2; \\ a_1 & \text{if } ez_i = ez_3; \\ a_1 \text{ or } a_2 & \text{otherwise.} \end{cases} \quad (5.4)$$

where ez_i is the last observation used to obtain eb_i . So, we have that $v^{e\pi^*}(s_1s_3) = v^{e\pi^*}(s_1s_2) = 1$.

However, in the original model $\overline{\mathcal{M}}$ the maximum value we can obtain for $\tau(s_1s_3)$ is $v^{\pi^*}(s_3) = 0.7$.

Thus we have that $v^{e\pi^*} \neq v^{\pi^*}$. □

Intuitively, an expansion with condition 5' does not prevent generating an equivalent model where the dynamics are the same, but information differ. In models with knowledge of the state, the expansion method must ensure that the agent has access to the same information available in the trajectories of the original system.

5.4 Discussion

When tackling real world problems with autonomous agents it is common to face settings that are naturally described as non-Markovian (i.e., relying on the past) and partially observable. In this chapter we presented an expansion method for extending p.o. non-Markovian reward decision processes, so as to obtain an equivalent POMDP, where a Markovian agent can find an optimal policy that is guaranteed to be optimal for the original non-Markovian model. We also provided proof that naive expansions from existing methods for fully observable models might find solutions that are not applicable in the original problem. Note that, in this work we considered the setting in which the observation function is deterministic to simplify the presentation. Nonetheless, a counterexample for the naive extension in this setting is a counterexample for the general case. Our work in this chapter provides theoretical ground for research lines solving complex instructions, such as complex temporal logic formulas, through RL in environments with imperfect knowledge of the state of the system. This kind of environments are empirically evaluated in Chapters 6 and 7. Future work can also explore how

EPOMDPs and EMGs (Chapter 4 can be effectively combined in multi-agent, partially-observable settings.

Part II

Drivers of Systematic Generalisation in Deep Reinforcement Learning

Chapter 6

Emergence of Grounded Systematic Generalisation with Abstract Operators

If I learned to avoid grass, shouldn't I be at least competent at avoiding lava? Chapter based on León et al. (2022a).

Systematic generalisation is a desirable ability towards reinforcement learning agents that operate in zero-shot, i.e., unseen, risky environments. For example, we can train a non-waterproof robot to walk while avoiding some innocuous object like grass without needing to worry about failures through learning to master object avoidance. In parallel, we can also *ground* the meaning of water, i.e., the dangerous thing to avoid, without requiring a direct interaction with it. Then, if the agent generalises systematically, it will be perfectly capable of avoiding water without ever requiring explicit training on the task.

In order to progress towards agents that execute zero-shot (i.e., unseen), instructions it is desirable to work with frameworks that facilitate the emergence of systematic learning in DRL. As anticipated

in Chapter 3, previous studies have shown that DRL agents can exhibit some degree of systematic generalisation of natural language instructions (Yu et al. 2018, Hill et al. 2021). Yet, there exists substantial evidence that DRL agents can learn systematically only when the right drivers are present (Lake 2019, Mao et al. 2019). Unfortunately, the focus on learning from natural instructions has hindered the study of drivers for systematicity with logic operators, e.g., negation or disjunction. Only Hill et al. (2020) provides evidence of systematic generalisation in DRL agents when facing negated zero-shot instructions. Notably, that previous work suggested that systematic learning can emerge from as few as six instructions when handling positive tasks, but that a much larger number of instructions is a strong requirement when learning to generalise abstract operators such as negation. This constraint would be a strong burden for DRL agents, meaning they could only generalise abstract concepts when rich and diverse environments and data are available.

In this chapter we give a new answer to the question: *can systematic learning emerge in DRL agents with abstract logic operators, e.g., negation, as early as without them?*. That is, can systematic learning emerge when learning from a limited variety (~ 6) of examples when working with negated commands? We provide empirical evidence that a DRL agent can generalise abstract operators – including negation – when learning from as few as six training instructions. In contrast, the previous state of the art stated that ~ 100 different instructions are always required (Hill et al. 2020). We demonstrate that the architecture of the convolutional layers (c.l.) – a background feature in previous literature – can be key on how agents overfit to the training tasks. We also bridge the literature of formal methods and systematic generalisation in DRL by introducing a novel neuro-symbolic framework where an agent with a single neural network can execute any complex zero-shot instruction in that language, while working in unseen maps of that domain. In this framework, the formal language helps neural-based agents that target zero-shot tasks, while the symbolic module facilitates generalisation to longer-than-training instructions.

The remaining of this chapter is structured as follows: Section 6.1 presents the formal language used to provide instructions, our neuro-symbolic framework, and the differences between convolutional architectures that we found to be key for systematic learning. Section 6.2 details the experimental settings and results obtained towards systematic generalisation. Finally, Section 6.3 reports a concluding

discussion on our work.

6.1 Solving Zero-Shot Instructions in Task Temporal Logic

Evaluating the emergence of systematic generalisation. We aim to assess whether a DRL agent is able to generalise systematically from temporal logic (TL) instructions. Consequently, *we do not focus on the raw reward collected by the agent*, but rather on whether the agent is correctly following new instructions. In line with previous literature (Hill et al. 2020, Icarte et al. 2019, Kuo et al. 2020), we work with two popular visual-based gridworld settings – MiniGrid (Chevalier-Boisvert et al. 2018) and the Minecraft-inspired environment (Andreas et al. 2017) – that involve procedurally-generated maps where agents need to fulfill reachability goals. In our experiments agents are trained with instructions such as “get gold” (reach gold) or “get something different from mud” (i.e., reach an object that is not mud). Then, we evaluate the agents with previously-unseen instructions, e.g., “get something different from iron, then get iron”. It is clear that a learning agent achieving significantly better rewards than a random walker does not imply systematic learning, e.g., the agent could accumulate larger rewards just by navigating better than a random walker and reaching all the objects promptly. For this reason, after training, we evaluate generalisation in settings called binary choice maps (BCMs). In these test maps, agents are evaluated with zero-shot reliable instructions, i.e., unseen *reliable* instructions that point to objects granting a positive reward, as well as zero-shot *deceptive* instructions, i.e., instructions pointing to objects granting a penalisation – a negative reward –. Since during training we always provide reliable instructions, agents that show systematic learning should do better than chance with zero-shot reliable instructions, while worse than a random agent when working with deceptive instructions.

This section presents our framework to tackle complex (multi-task) zero-shot temporal specifications. It relies on the three components detailed below: 1) a formal language, whose atoms are designed to facilitate the systematic learning of a neural-based module; 2) a symbolic module, whose role is to decompose specifications into sequences of simpler tasks, and provide the reward signal for the neural module on the current task to execute; 3) a neural module consisting of a DRL agent aimed to

learn systematically from the tasks fed by the symbolic module.

6.1.1 Task Temporal Logic

We start by defining a novel TL language whose atomic components are used to train a DRL algorithm. Task temporal logic (TTL) is an expressive, learning-oriented TL language interpreted over finite traces, i.e., over finite episodes. Given a set AT of atomic tasks $\alpha, \beta, \dots$, the syntax of TTL is defined as follows:

Definition 6.1 (TTL). *Every formula ϕ in TTL is built from atomic tasks $\alpha \in AT$ by using negation “ $\sim$ ” (on atoms only), sequential composition “ $;$ ” and non-deterministic choice of atoms “ $\vee$ ” and formulae “ $\cup$ ”:*

$$T ::= \alpha \mid \alpha \sim \mid \alpha \vee \alpha$$

$$\phi ::= T \mid \phi; \phi \mid \phi \cup \phi$$

Intuitively, an atomic task α corresponds to a reachability goal in the TL literature, in the sense that the fulfilment condition associated with α will eventually hold. Task $\alpha \sim$ encapsulates a special form of negation; informally it means that something different from α will eventually hold. Choosing this form of negation allows us to contrast our results with those of previous studies of this operator in natural language (Hill et al. 2020). Note that the negation symbol is intentionally positioned after the atomic task. We found that this feature helps during training with visual instructions, as it forces the neural module to process the same number of symbols to distinguish negative from positive tasks (we refer to Section 6.1.3 for further detail). Formulae $\phi; \phi'$ intuitively express that ϕ' follows ϕ in sequential order; whereas $\phi \cup \phi'$ means that either ϕ or ϕ' holds. We then also introduce an abbreviation for the concurrent (non-sequential) composition $\cap$ as follows: $\phi \cap \phi' \equiv (\phi; \phi') \cup (\phi'; \phi)$. We say that our language TTL is *learning-oriented* as its logical operators and their positions in formulae are so chosen as to help the training process of the learning agent.

TTL is interpreted over finite traces λ , where $|\lambda|$ denotes its length. We denote time steps, i.e., instants,

on the trace as $\lambda[j]$, for $0 \leq j < |\lambda|$; whereas $\lambda[i, j]$ is the (sub)trace between instants i and j . In order to define the satisfaction relation for TTL, we associate with every atomic task α an atomic proposition p_α , which represents α 's fulfilment condition. Note again that, in this context, α is a *reachability goal*, typically expressed in TL as an *eventuality* $\diamond p_\alpha$. Then, a *model* is a tuple $\mathcal{N} = \langle \mathcal{M}, L \rangle$, where $\mathcal{M}$ is a POMDP, and $L : S \rightarrow 2^{AP}$ is a labelling of states in S with truth evaluations of atoms p_α in some set AP of atoms.

Definition 6.2 (Satisfaction). *Let $\mathcal{N}$ be a model and λ a finite trace. We define the satisfaction relation $\models$ for tasks T and formulae ϕ on trace λ inductively as follows:*

$$\begin{aligned}
(\mathcal{N}, \lambda) \models \alpha & \quad \text{iff for some } 0 \leq j < |\lambda|, p_\alpha \in L(\lambda[j]) \\
(\mathcal{N}, \lambda) \models \alpha \sim & \quad \text{iff for some } 0 \leq j < |\lambda|, \text{ for some } q \neq p_\alpha, q \in L(\lambda[j]) \text{ and } p_\alpha \notin L(\lambda[j]) \\
(\mathcal{N}, \lambda) \models \alpha \vee \alpha' & \quad \text{iff } (\mathcal{N}, \lambda) \models \alpha \text{ or } (\mathcal{N}, \lambda) \models \alpha' \\
(\mathcal{N}, \lambda) \models \phi; \phi' & \quad \text{iff for some } 0 \leq j < |\lambda|, (\mathcal{N}, \lambda[0, j]) \models \phi \text{ and } (\mathcal{N}, \lambda[j+1, |\lambda|-1]) \models \phi' \\
(\mathcal{N}, \lambda) \models \phi \cup \phi' & \quad \text{iff } (\mathcal{N}, \lambda) \models \phi \text{ or } (\mathcal{N}, \lambda) \models \phi'
\end{aligned}$$

By Definition 7.2, an atomic task α indeed corresponds to a formula $\diamond p_\alpha$ of temporal logic, where p_α is the fulfilment condition associated with α . It can be immediately inferred that TTL is a fragment of the widely-used Linear-time Temporal Logic over finite traces (LTL_f) (De Giacomo & Vardi 2013). We provide a translation of TTL into LTL_f and the corresponding proof of truth preservation in Sec. A.3 in the appendix. To better understand the language we present a toy example.

Example 3. *The specification “get something different from grass, then collect grass and later use either the workbench or the toolshed” can be expressed in TTL as:*

$$\phi \triangleq (\text{grass} \sim); \text{grass}; (\text{workbench} \vee \text{toolshed})$$

TTL is designed to be expressive enough to encapsulate both the tasks described in Andreas et al. (2017), a popular benchmark in the RL-TL literature (Camacho et al. 2019, De Giacomo et al. 2019, León & Belardinelli 2020), and the negated instructions from studies about negation in natural language (Hill et al. 2020), i.e., encapsulating “not wood” as “get something different from wood”.

Algorithm 2 The symbolic module (SM)

```

1: Input: Instruction  $\phi$ 
2: Generate the accepted sequences of tasks  $\mathcal{K} \leftarrow \mathcal{E}(\phi)$ 
3: Retrieve the current observation:  $z$ 
4: Get the true proposition:  $p \leftarrow \mathcal{L}_I(z)$ 
5: Get the first task:  $T \leftarrow \mathcal{P}(\mathcal{K}, p)$ 
6: repeat
7:   Generate the extended observation:  $z^{ext} \leftarrow z + T$ 
8:   Get the next action:  $a \leftarrow \text{NM}(z^{ext})$ 
9:   Execute  $a$ , new observation  $z'$ 
10:  New true proposition:  $p' \leftarrow \mathcal{L}_I(z')$ 
11:  Provide the internal reward:  $\text{NM} \leftarrow R_I(p')$ 
12:  if  $p' == p_\alpha$  for  $\alpha \in T$  then
13:    Update  $\mathcal{K}, T' \leftarrow \mathcal{P}(\mathcal{K}, p')$ 
14: until  $\mathcal{K} == \emptyset$  or time limit

```

While TTL allows for more complex instructions than those studied in Hill et al. (2020), it is less expressive than other languages used in state-of-the-art methods combining temporal logic and reinforcement learning (Alshiekh et al. 2018, Yuan et al. 2019, Icarte et al. 2018, De Giacomo et al. 2019). Nevertheless, differently from those works, in this chapter we design an agent that is able to work with high-dimensional inputs and “abstracts” logical operators.

6.1.2 The Symbolic Module

Given an instruction ϕ in TTL, the first task of the symbolic module (SM) is to decompose ϕ into sequences of atomic tasks for the neural module (NM). Algorithm 2 details the functioning of the SM. Intuitively, the SM reduces the problem into a progression of POMDPs, which is a well-known procedure in the literature on RL with TL specifications (Brafman et al. 2018, Toro Icarte et al. 2018).

In particular, a task extractor $\mathcal{E}$ (Algorithm 3) transforms ϕ into the set $\mathcal{K}$ of all sequences of tasks T that satisfy ϕ . As standard in the literature (Andreas et al. 2017, Kuo et al. 2020), we assume that the SM has access to an internal labelling function (a.k.a. event detector¹) $\mathcal{L}_I : Z \rightarrow 2^{AP}$, which maps the agent’s observations into sets of atoms in AP . Note that $\mathcal{L}_I$ might differ from the labelling $\mathcal{L}$ originally defined in Sec. 6.1.1, since $\mathcal{L}$ is defined on states. For our approach to be effective, observations have

¹Note that our assumptions are not stronger than those in any typical RL work (Sutton & Barto 2018) which assume access to a reward function that maps states to rewards.

Algorithm 3 Extractor function that obtains the list-of-lists with the possible sequences of tasks.

```

1: Function  $\mathcal{E}$ , Input:  $\phi$ 
2: Initialize the list  $\mathcal{K}$  with an empty sequence of tasks  $Seq_0$ 
3: for each atomic task  $T \in \phi$  do
4:   if  $T$  is not a non-deterministic choice then
5:     for all  $Seq \in \mathcal{K}$  do
6:        $Seq.append(T)$ 
7:   else
8:     Save the first element:  $FC \leftarrow T[0]$ 
9:     Save the last element:  $SC \leftarrow T[-1]$ 
10:     $Nseqs \leftarrow length(\mathcal{K})$ 
11:    for all  $Seq \in \mathcal{K}$  do
12:      Generate a clone:  $Seq' \leftarrow Seq$ 
13:       $\mathcal{K}.append(Seq')$ 
14:      for  $i$  in range( $length(\mathcal{K})$ ) do
15:        if  $i < Nseqs$  then
16:           $\mathcal{K}[i].append(FC)$ 
17:        else
18:           $\mathcal{K}[i].append(SC)$ 
19: Output:  $\mathcal{K}$ 

```

to contain enough information to decide the truth of atoms, so that both the agent’s internal and the model’s overall labelling functions remain aligned. Given $\mathcal{L}_I$ and the sequences of tasks in $\mathcal{K}$, a progression function $\mathcal{P}$ described in Algorithm 4 selects the next task from $\mathcal{K}$ that the NM has to solve. Once a task is solved, $\mathcal{P}$ updates $\mathcal{K}$ and selects the next task. This process is repeated until ϕ is satisfied or the time limit is reached.

When forwarding the current task T to the NM, the SM expands observation z with the subformula corresponding to T . In the current context, z is a square visual input with a fixed perspective. This is expanded either by adding extra rows to z containing T (see Figure 6.3 right) or providing T in a separated “language” channel (see Figure 6.4). During the episode the SM also provides the reward signal that guides the NM to solve T :

$$R(p_t) = \begin{cases} d & \text{if } p_t = \emptyset, \text{ where } d \in \mathbb{R} \text{ and } d < 0; \\ i & \text{if } p_t = p_\alpha, \text{ for } \alpha \text{ occurring in } T \text{ and } T \neq \alpha \sim, \text{ or if } p_t \neq p_\alpha, \text{ for } \alpha \text{ occurring in} \\ & T \text{ and } T \equiv \alpha \sim, \text{ where } i \in \mathbb{R} \text{ and } i \geq 0; \\ c & \text{otherwise, where } c \in \mathbb{R} \text{ and } c \ll d. \end{cases} \quad (6.1)$$

Algorithm 4 Progression function that returns the next task to be solved.

```

1: Function  $\mathcal{P}$ , Input:  $\mathcal{K}, p$ 
2: if  $p \neq \emptyset$  then
3:   for all  $Seq \in \mathcal{K}$  do
4:     if  $p$  fulfills  $Seq[0]$  then
5:        $Seq.pop(Seq[0])$ 
6:     else
7:        $\mathcal{K}.pop(Seq)$ 
8:  $T \leftarrow \emptyset$ 
9: if  $\mathcal{K}$  is not  $\emptyset$  then
10:   Select the head  $T$  of a sequence in  $\mathcal{K}$ :
11:   if  $T \equiv \alpha$  then
12:     //We have an atomic prop. Thus, we are aiming to form a non-deterministic choice
13:     for all  $Seq \in \mathcal{K}$  do
14:       if  $T == Seq[0]$  then
15:         continue
16:       else if  $Seq[0] \equiv \alpha$  then
17:          $T \leftarrow T \vee Seq[0]$ 
18:       return  $T$ 
19: Output:  $T$ 

```

where $\mathcal{L}_I(z_t) = p_t$ for any time step t . Intuitively, the NM is rewarded when the event detector $\mathcal{L}_I$ fires a signal that satisfies the current task, and penalised when $\mathcal{L}_I$ signals an event not related to T . A small penalisation is also given at each time step to induce the agent to promptly solve T . We now illustrate the inner working of the SM.

Example 4. Consider one of the complex specifications used in Sec. 6.2: $\phi_1 = ((\text{wood}; \text{grass}) \cup (\text{iron}; \text{axe})); \text{workbench}; (\text{toolshed} \sim)$. Given ϕ_1 , the task extractor $\mathcal{E}$ outputs set $\mathcal{K} = \{ \langle [\text{wood}], [\text{grass}] \rangle, [\text{workbench}], [\text{toolshed} \sim] \rangle, \langle [\text{iron}], [\text{axe}], [\text{workbench}], [\text{toolshed} \sim] \rangle \}$ with two available sequences of tasks. Set $\mathcal{K}$ is forwarded to the progression function $\mathcal{P}$ and since we have two lists with different initial elements, both positive, $\mathcal{P}$ selects $[\text{wood} \vee \text{iron}]$ as task T to extend the next observations of the NM. The NM iterates with the environment using the extended observations as input. At each iteration, functions $\mathcal{L}_I$ and R respectively evaluate the progression of T and reward the NM until T is satisfied. In our example, if the NM got wood, the second sequence and $[\text{wood}]$ are discarded and the next task becomes $[\text{grass}]$. This process is repeated until $\mathcal{K}$ is empty or the time limit is reached.

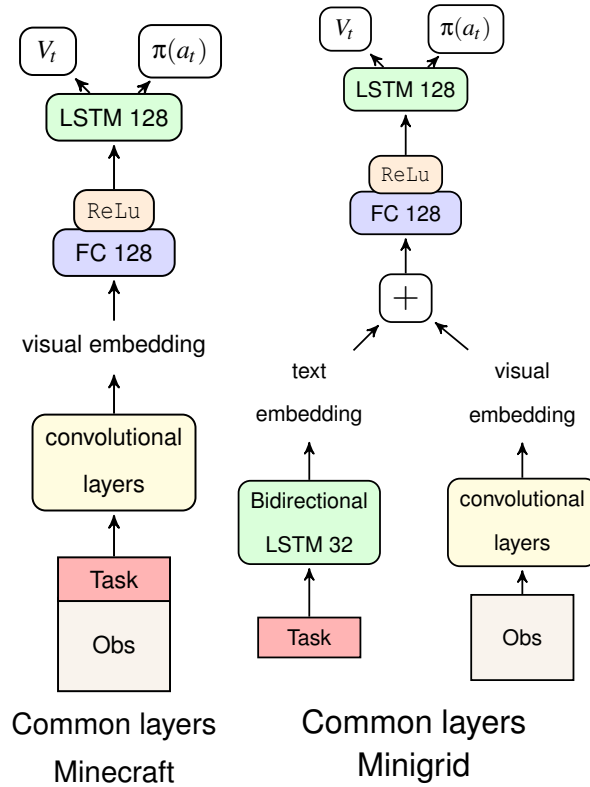


Figure 6.1: Common features of the neural network architectures evaluated in Section 6.2. All the variants tested in the Minecraft-inspired environment follow the architecture depicted on the left. The one on the right is the structure employed in Minigrid. $\pi(a_t)$ and V_t refer to the output actor and critic layers respectively Mnih et al. (2016).

6.1.3 The Neural Module

In this last part of the section we detail insights on the NM and novel insights we faced when training a neuro-symbolic agent that systematically generalises the negation and non-deterministic choice operators.

Neural Module. Our NM consists of a DRL algorithm A2C, that is trained in procedurally generated maps. We employ a fixed perspective which is known to help with generalisation (Mao et al. 2019, Hill et al. 2020). Note that the different agents evaluated in our experiment section differ solely on their CNN architecture.

CNN architectures. We study architectures with convolutional layers followed by a fully connected (f.c.) layer (Goodfellow et al. 2016) and an LSTM (Gers et al. 2000)). In preliminary studies, we ob-

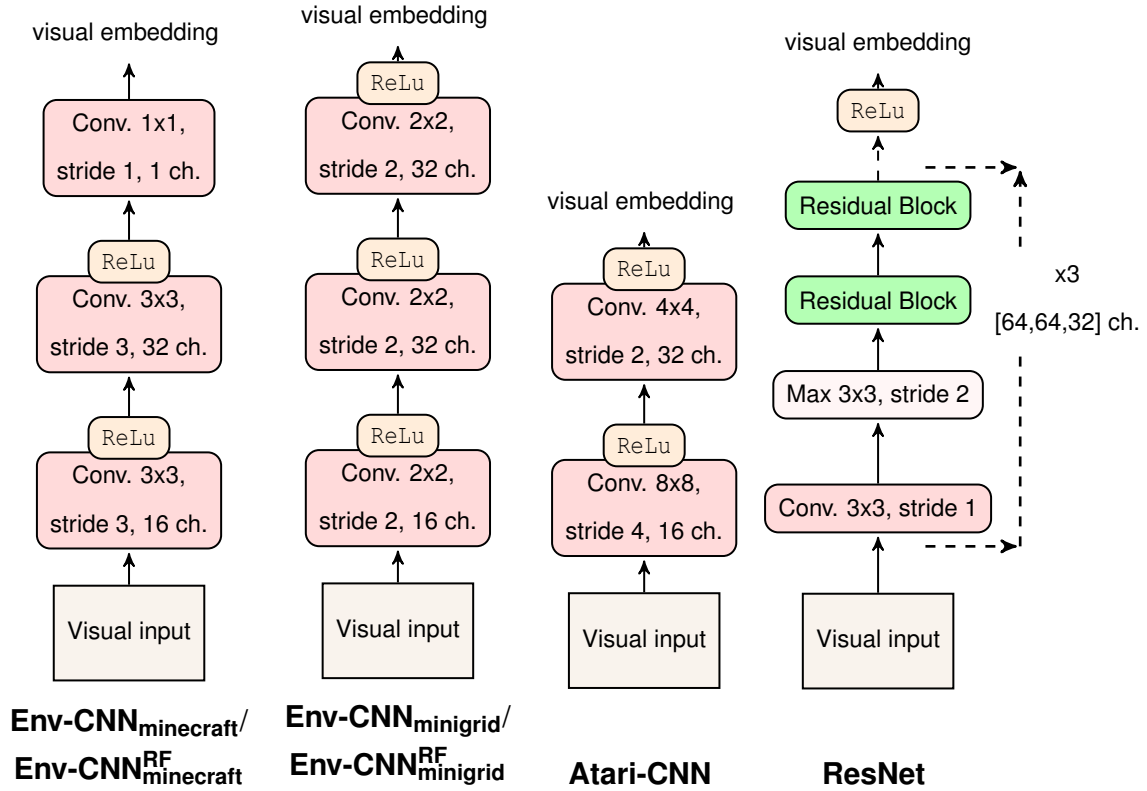


Figure 6.2: The different convolution architectures explored in this chapter. The different number of channels in the last layer of the Env-CNN architectures is done to provide evidence that such feature does not affect how the agent generalises.

served that agents with different f.c. and recurrent layer configurations exhibit similar generalisation abilities. However, as illustrated in Sec. 6.2, different convolutional configurations yield dramatically different results. Intuitively, we find that systematic generalisation emerges when the visual encoders induce object-oriented representations early enough, i.e., that the visual encoder learns to produce such object-centric embeddings before the agent starts to overfit. In our experiments with grid-world visual environments we explored different architectures exploiting the knowledge of the tile resolution of each environment, and detected three “types” of CNNs according to how much architectures *align* with their environment. The first type are the *weakly aligned* (WA) architectures, these architectures have kernel and stride sizes that are factors of the tile resolution of the environment. The second type are *strongly aligned* (SA) architectures, these are networks that fulfill both the WA conditions and where the two first dimensions (length and width) of their visual embedding correspond to the number of input tiles in the original observation. Note that the number of channels of the output, i.e., the last dimension of the embedding, does not influence the alignment. Last, we call *not aligned* (NA) CNNs those architectures that are not WA or SA. When aiming at systematic learning

from six training instructions, we find that only SA networks consistently allow the emergence of systematic generalisation. Moreover, we find that the difference in alignment already between WA and NA has more impact than the depth of the network. Further, we show that a shallow SA CNN with random weights can achieve stronger generalisation than a trained NA ResNet. All of this contrasts an extended belief that the “depth” of the convolutional layer is the most important feature in visual encoders towards generalisation (Cobbe et al. 2019).

Negation. When learning positive and negative instances concurrently using visual instructions, we noticed that the position of the negation operator impacted on the agent’s behavior in training. Specifically, placing the negation symbol after the negated object helps the agent to learn both types of instances concurrently, while, when placed otherwise, the agent focus only on the positive tasks first, which can lead to saddle points. We believe that this result is not related to linguistic processing. Note that all of our architectures in Minecraft-inspired process the specification as a whole, as part of the observational input processed by the convolutional layer. Thus, the order is not important from the point of view of temporal processing. We believe the difference was caused because affirmative formulas are represented with one symbol, e.g., “get wood” is represented as “wood”; while negation is represented by two symbols, e.g., “get an object different from wood” as “wood $\sim$ ”. By placing the negation operator after the object we force the neural network to “observe” the first two inputs to differentiate affirmative tasks from negative ones. An alternative approach that could have a similar effect is adding an explicitly positive operator so that both negative and positive formulas have the same length.

Non-deterministic Choice. Training the agent to effectively learn non-deterministic choice $\cup$ required episodes where only one of the two disjuncts of the non-deterministic choice was present, as well as some other episodes with the two objects present simultaneously. Intuitively, if we have the instruction “get axe or get gold”, there should be training episodes where gold is present in the map, but there are no axes, and similarly, other maps where there are axes, but there is not gold. This prevents the agent from biasing with the object that appears either in the first or second position of the instruction. Additionally, the agent should also interact with episodes where both objects are present.

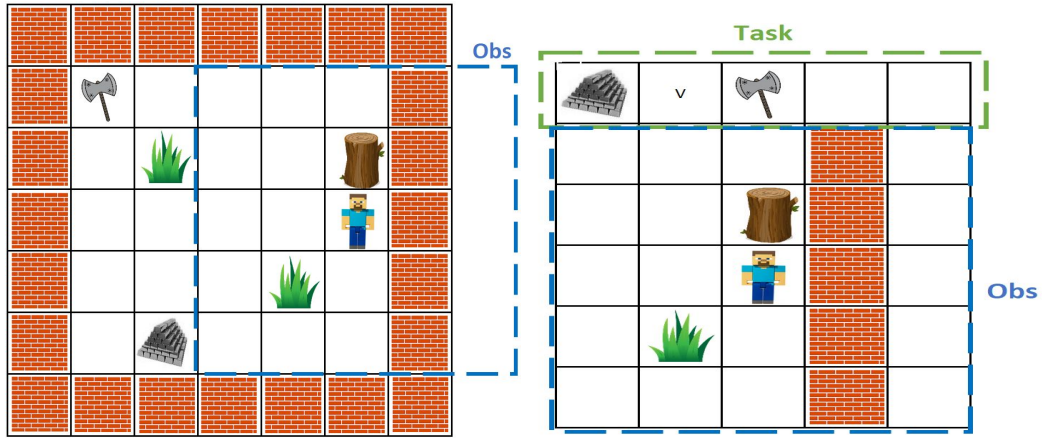


Figure 6.3: **Left.** Procedurally generated map in the Minecraft-inspired environment with its corresponding extended observation (**right**) when the agent’s task is to reach either iron or an axe (top row). Tasks are specified in the top row of the extended observation in this setting by depicting the object itself.

Otherwise, the policy network of the agent will allocate a similarly high probability to go after the two objects, instead of to the one that is closer or “safer”, i.e, far from non-desired objects.

6.2 Experiments

In this section we first describe the characteristics and details of our experimental analysis. Later we include and discuss the results on the ability of agents to achieve systematic generalisation on abstract operators.

6.2.1 Experiment Settings.

We perform our empirical study on two popular procedural environments. The first one (Figure 6.3) is designed along the lines of a Minecraft-inspired (Minecraft for short) setting widely used in previous works combining formal methods and RL (Andreas et al. 2017). This Minecraft environment consist of a gridworld with typical Minecraft objects and tools, e.g. wood, stone or workbench, that the agent can interact with. The action set consists of 4 individual actions: *Up*, *Down*, *Left*, *Right*, that move the agent one tile in the given direction. We adapt the environment originally introduced in Andreas et al. (2017) to match the pixel-based 2D maps from Hill et al. (2020) (the previous study about

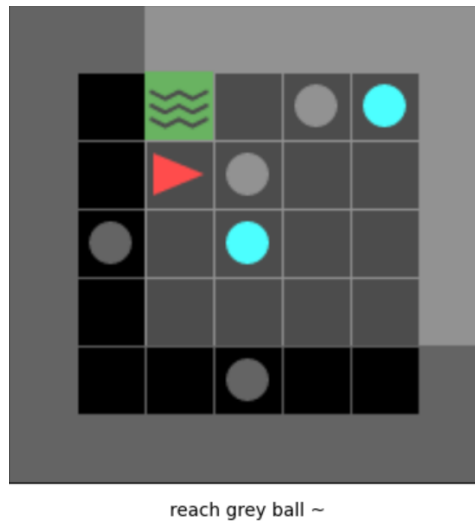


Figure 6.4: Procedurally generated map in MiniGrid where the task is to reach an object different from a grey ball, i.e., “*not* a grey ball”. Here instructions are given on a separated channel and objects are combinations of shapes and colors.

negation). Here each tile has a resolution of $9 \times 9 \times 1$. The DRL algorithm receives observations extended with visual instructions depicting the target objects (Figure 6.3 right). MiniGrid (Chevalier-Boisvert et al. 2018) is our second experimental environment, depicted in Figure 6.4 right. Here each tile has a resolution of $8 \times 8 \times 3$, and the action set is *Move forward*, *Turn left*, *Turn right*. In MiniGrid instructions are given through a separated text channel.

CNN architectures. For each setting we test four different CNN configurations depicted in Figure 6.2. The first two are popular architectures from previous literature that we use as baselines: *Atari-CNN* (Mnih et al. 2015, 2016) which is NA with Minecraft and WA with MiniGrid, and *ResNet* (Espenholt et al. 2018, Hill et al. 2020) WA with Minecraft and NA with MiniGrid. We contrast these networks with different architectures designed to be SA with each environment and that we refer to as Env-CNN. Last, for each environment we also test the performance of the same Env-CNNs, but when the convolutional layers are frozen during training, i.e., these layers always use a set of randomly selected parameters. As anticipated in Section 6.1.3. We refer to these networks as Env-CNN^{RF}. Note that we intentionally use a significantly different number of channels in the last layers with Env-CNN_{minecraft} and Env-CNN_{minigrid}. We do this to evidence that the number of channels in the visual embedding does not affect the ability of SA architectures to generalise training instructions. Common schemes of all architectures in the different experimental setting are given in Figure 6.1.

Procedural Maps Training maps are procedurally-generated with a random number of objects that ranges from 2 to 8, there are no restrictions on the number of objects on a training map that can be the same as long as one of the objects solves the current task. If the agent moves against a wall, its position remains the same. Note that moving against walls is discouraged by the internal reward function that provides a negative reward at each timestep if the agent has not solved the current task.

Instruction generation. In every experimental setting we refer to the global set of objects as $\mathcal{X}$, which is split into training ($\mathcal{X}_1$), validation ($\mathcal{X}_2$) and test sets ($\mathcal{X}_3$), where $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$ are pairwise disjoint. $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, $\mathcal{X}_1 \cap \mathcal{X}_3 = \emptyset$, and $\mathcal{X}_2 \cap \mathcal{X}_3 = \emptyset$. We contrast the results of learning with training sets of different size that we refer to as *small* ($|\mathcal{X}_1| = 6$) and *large* ($|\mathcal{X}_1| = 20$). In Minecraft, where instructions are visual, agents are trained with objects from $\mathcal{X}_1$ only. In MiniGrid, where agents require to ground words to objects, we train the agents with positive instructions from $\mathcal{X}$, while instructions concerning the negation or disjunction operators refer only to objects in $\mathcal{X}_1$. Agents are trained with atomic tasks T . Tasks are procedurally generated with objects from the corresponding set. For instance, with the “small” setting we have six different objects, meaning that agents are learning the negation operator from six training instructions. At each episode the agent navigates receiving rewards according to T . The episode ends when the agent receives a positive reward (meaning that T is satisfied) or the time limit is reached. During training we periodically evaluate the performance on the corresponding validation set $\mathcal{X}_2$, that we use to select the best learning weights in each run, i.e., to perform early stopping when necessary. Figure 6.5 shows the different training and evaluation plots. Further details about hyperparameters and hardware are given in Appendix A.1.

Rewards. In both settings the agents received a penalisation of -1 when reaching an object that does not satisfy the task, and +1 when the opposite occurs. We explored different penalisation values per time-step to stop the agents from local-optima policies, e.g., not reaching any object, while still making beneficial avoiding undesired objects when possible. We did a grid-search in the range $[-0.01, -0.2]$ and achieved the best results with -0.1 in Minecraft and -0.05 in MiniGrid.

Object-set generation. In the Minecraft-inspired environment, we procedurally generate objects as a random matrix of 9x9 values. Small and large training sets use 6 to 20 procedural objects as described above. For validation and test sets we use 6 objects respectively. In MiniGrid, objects are not procedural values but compositions of colors, e.g., red, and shapes, e.g., key. Here the small set uses 2 shapes and 3 colors while training with negative instructions and disjunction. In validation we use an additional shape combined with training colors. In test we use 2 new colors, the validation shape and a new test shape. In the large setting, training has 4 shapes and 5 colors, an additional shape with training colors for validation, and 2 new colors with a test shape and the validation shape in test.

6.2.2 Empirical Results

Results about Systematic Learning. Table 6.1 shows the results in a particular set that we call binary choice maps (BCMs). These maps have only two objects (except in “disjunction 2 choices”, with four), one granting a positive reward ($i = +1$) the other a penalisation ($c = -1$) (see equation 6.1.2 above). In “disj. 2” two objects penalise, and two provide a reward. Since agents are always trained with reliable instructions pointing to objects that grant a positive reward, an agent that is correctly generalising to zero-shot instructions should collect significantly higher rewards with reliable instructions than with deceptive ones (i.e., the later instructions are pointing to objects that grant negative rewards). Based on previous literature (Hill et al. 2020), agents should accumulate roughly two times higher rewards with reliable instructions than when given deceptive ones. Nevertheless, the larger the difference, the stronger systematic learning.

The first noticeable pattern from Table 6.1 is that only architectures with some form of alignment (WA and SA) give evidence of systematic learning when no more than six training instructions are available. NA architectures show minimal to none generalisation (e.g., better performance with deceptive instructions than with reliable ones) in all the “small” settings. With 20 objects, NA architectures also fail when instructions are visual (Minecraft), while struggling with the disjunction operator. When instructions come in a dedicated channel (Minigrid), architectures with some alignment (WA) give some evidence of systematic generalisation. Yet, when instructions are visual, WA CNNs only learn

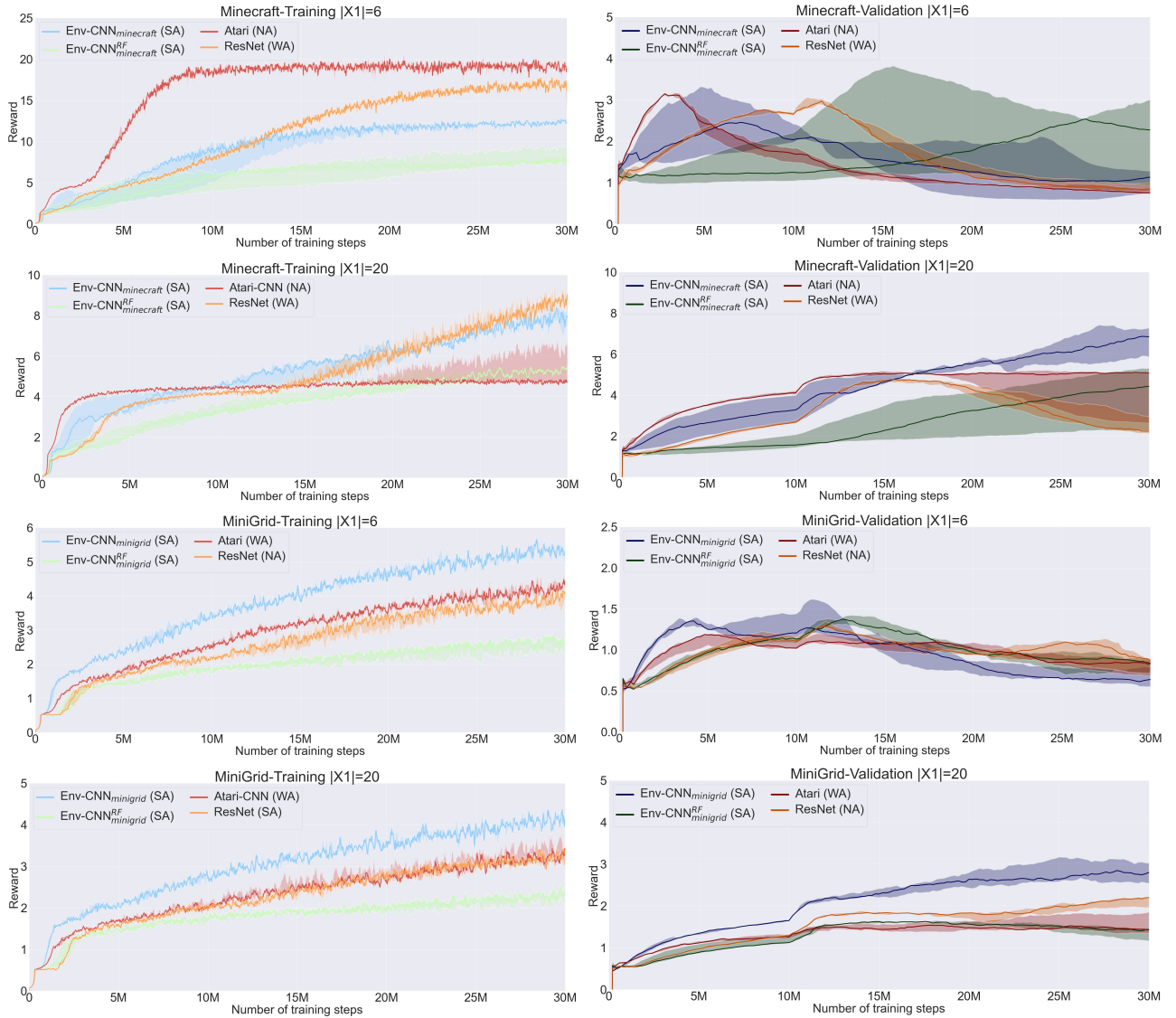


Figure 6.5: Train and validation plots with the different population sizes and 5 independent runs. Light colors refer to performance with training objects, whereas dark refer to validation objects. Lines refer to the median while shaded areas represent the 25-75 quartile difference.

Table 6.1: Results from 500 BCMs in Minecraft (MC) and MiniGrid (MG) with negated instructions and the most challenging disjunction-based tasks. In “Reliable” higher is better while in “Deceptive” lower is better. Rewards are normalised so that 1.0 is the performance of a random agent. Agents show a stronger systematic generalisation the bigger the difference between respective reliable and deceptive instructions. Pos. and neg. labels refer to atomic positive or negative instruction respectively, pointing to one of the two objects present. In disj. 2nd the instructions refers to two objects, but only the second is present. Disj. 2 points to two objects present in a map with 4 items. Results are bolded where both rewards with reliable instructions is greater than 2.0, i.e., twice the rewards of a random agent, and less than 1.0 with deceptive instructions, i.e., the agent successfully exhibits a policy that avoids the rewarding object when the instruction says so.

<i>MC</i>	$ X_1 $	Atari-CNN (NA)		ResNet (WA)		Env-CNN _{minecraft} (SA)		Env-CNN _{minecraft} ^{RF} (SA)	
Instr.		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Pos.	6	3.89 (0.1)	3.92(0.1)	3.72(0.5)	3.42(0.8)	3.25 (1.9)	0.26 (1.9)	3.95 (1.8)	0.74 (0.7)
	20	4.14(0.1)	4.16(0.1)	3.30(1.8)	1.10(1.4)	5.43 (0.2)	0.11 (0.0)	3.95(1.8)	1.10(1.2)
Neg.	6	3.82 (0.2)	3.81(0.1)	3.52(0.5)	3.49(0.7)	2.20 (1.3)	0.44 (0.4)	2.23(1.9)	1.04(0.8)
	20	3.94(0.1)	3.97(0.1)	3.09(1.7)	1.26(0.9)	4.40 (0.7)	0.14 (0.0)	2.25 (1.8)	0.94 (1.2)
Disj.	6	2.56 (0.2)	2.59(0.6)	2.29(0.1)	2.51(0.7)	2.23 (4.8)	0.57 (0.3)	2.54(0.7)	1.37(0.1)
	20	2.63 (0.2)	2.68(0.6)	2.16(0.1)	2.23(0.7)	3.53 (4.8)	0.44 (0.3)	2.82 (0.7)	1.07 (0.9)

<i>MG</i>	$ X_1 $	Atari-CNN (WA)		ResNet (NA)		Env-CNN _{minigrid} (SA)		Env-CNN _{minigrid} ^{RF} (SA)	
Instr.		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Neg.	6	2.83 (1.0)	0.44 (0.3)	3.02(1.4)	2.97(1.4)	2.01 (1.3)	0.41 (0.3)	1.61 (0.2)	0.75 (0.2)
	20	2.77 (0.8)	0.34 (0.1)	5.05 (1.3)	0.20 (0.1)	5.03 (1.7)	0.16 (0.0)	2.16 (0.2)	0.98 (0.2)
Disj.	6	4.21 (0.5)	0.84 (0.5)	4.02(0.4)	3.68(0.6)	2.67 (1.5)	0.79 (0.5)	4.06 (0.7)	0.63 (0.4)
	20	3.42(1.0)	1.14(0.5)	5.03(0.6)	1.75(0.8)	3.13 (0.5)	0.37 (0.1)	2.72(1.0)	1.58(0.4)
Disj.	6	2.83 (0.9)	0.90 (0.4)	3.52(0.8)	3.86(1.3)	1.68 (1.3)	0.75 (0.3)	2.81 (0.8)	0.88 (0.5)
	2 nd 20	2.11(1.4)	1.06(0.3)	2.60(0.7)	2.37(1.0)	1.81 (0.5)	0.50 (0.1)	1.91(0.8)	1.28(0.4)

to generalise positive and negative instructions, although they still fail to avoid the objects consistently in deceptive settings. Remarkably, architectures that are strongly aligned (SA) show better evidence of systematic generalisation even when the weights of the convolutional layers are not trained (Env-CNN^{RF}), outperforming a Resnet that was trained and used early-stopping, but whose architecture was NA. *This is a strong evidence that a critical feature for generalisation is that the convolutional layers produce an object-centric visual embedding*, which in our study the SA networks automatically produce, independently of their weights. Moreover, the Env-CNN (the trained SA architecture) is the only configuration that systematically achieves a performance greater than 1.0 (the performance of a random agent) with reliable instructions, while getting less than 1.0 with deceptive instructions across all settings. That is, within our experiments, *SA CNNs is the only family of architectures that systematically learns to generalise under limited object diversity* and correctly infers not only the desired

Table 6.2: Complete results in BCMs with disjunctions. Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

MC	$ \mathcal{X}_1 $	Atari-CNN (NA)		ResNet (WA)		Env-CNN _{minecraft} (SA)		Env-CNN _{minecraft} ^{RF} (SA)	
Instr.		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Disj. 1 st	6	3.39 (0.1)	2.12(0.0)	2.89(0.3)	2.04(0.1)	3.01 (1.4)	0.76 (0.6)	2.58 (1.8)	1.24 (0.5)
	20	3.60(0.0)	2.18(0.0)	2.67(1.2)	1.77(1.0)	5.10 (1.2)	0.47 (0.5)	2.93(1.6)	1.02(0.8)
Disj. 2 nd	6	3.25 (0.2)	2.13(0.0)	2.83(0.5)	2.02(0.2)	3.05 (1.3)	0.57 (0.5)	2.63 (1.7)	1.05 (0.5)
	20	3.60(0.1)	2.17(0.0)	2.89(1.3)	1.65(0.2)	4.91 (1.5)	0.53 (0.6)	2.96(1.9)	1.04(0.7)
Disj. 2	6	2.56 (0.0)	2.59(0.1)	2.29(0.3)	2.51(0.2)	2.23 (1.0)	0.57 (0.6)	2.54(0.7)	1.37(0.7)
	20	2.63 (0.0)	2.68(0.0)	2.16(0.5)	2.23(0.2)	3.53 (0.3)	0.44 (0.4)	2.82 (0.8)	1.07 (0.9)

MG	$ \mathcal{X}_1 $	Atari-CNN (WA)		ResNet (NA)		Env-CNN _{minigrid} (SA)		Env-CNN _{minigrid} ^{RF} (SA)	
Instr.		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Disj. 1 st	6	2.48 (0.6)	0.93 (0.3)	3.73(0.6)	3.64(1.3)	2.09 (1.2)	0.95 (0.6)	3.36 (0.9)	0.79 (0.5)
	20	2.99(0.86)	1.01(0.4)	6.14 (0.8)	1.49 (0.8)	2.61 (0.6)	0.39 (0.1)	2.06(0.8)	1.31(0.3)
Disj. 2 nd	6	2.83 (0.9)	0.90 (0.4)	3.52(0.8)	3.86(1.3)	1.68 (1.2)	0.75 (0.3)	2.81 (0.8)	0.88 (0.55)
	20	2.11(1.4)	1.06(0.3)	2.60(0.7)	2.37(1.0)	1.81 (0.5)	0.50 (0.1)	1.91(0.8)	1.28(0.4)
Disj. 2	6	4.21 (0.5)	0.84 (0.5)	4.02(0.4)	3.68(0.6)	2.67 (1.5)	0.79 (0.5)	4.06 (0.7)	0.63 (0.4)
	20	3.42(1.0)	1.14(0.5)	5.03 (0.6)	1.75 (0.8)	3.13 (0.5)	0.37 (0.1)	2.72(1.0)	1.58(0.4)

object/s, but also which undesired items should be avoided according to the zero-shot instruction.

We also highlight that in preliminary studies we explored additional overfitting-prevention methods, including dropout and autoencoders, to help NA networks. However, the results do not vary from the ones obtained by the train-validation-test set division (with early stopping) that we carry here and that provide evidence of the benefits of using architectures that are aligned with their environment.

Results on Disjunction. Table 6.2 shows the results of the different architectures when following reliable and deceptive zero-shot instructions with non-deterministic choices. Instructions labelled “Disjunction 1st choice” refer to those where only the first object within the instruction is present in the map. “Disjunction 2nd choice” is a similar setting with the second object. In “Disjunction 2 choices” both objects are present in the map. In line with earlier results we see that only SA architectures give evidence of systematic generalisation in every test.

Table 6.3: Results from 200 maps with test complex instructions, i.e, longer than training instructions composed only by zero-shot tasks. Steps refers to the number of actions per instruction needed by the agents. The highest value over all networks is bolded.

	Atari-CNN (NA)		ResNet (WA)		Env-CNN _{minecraft} (SA)		Env-CNN _{minecraft} ^{RF} (SA)	
$ \mathcal{X}_1 $	Reward	Steps	Reward	Steps	Reward	Steps	Reward	Steps
6	2.45 (0.25)	26.3 (5.0)	2.68(0.65)	49.3(28.9)	3.08 (1.49)	49.1(25.8)	2.74(1.78)	56.51(22.9)
20	2.99(0.21)	20.96 (1.50)	3.49(0.68)	27.4(5.2)	7.45 (3.36)	32.84(6.8)	5.49(1.21)	38.2(9.7)

Generalisation with TTL formulae. We last evaluate the functioning of the entire framework. Thus, we use the same agents trained with atomic tasks T when executing complex (multi-task) instructions given as TTL formulae that refer to zero-shot atomic tasks. Table 6.3 shows the average reward and number of actions needed in 200 maps executing complex instructions. We see that a stronger alignment yields better final performance (i.e., $SA > WA > NA$ accumulated reward).

Table 6.4 contains the five complex instructions that we use to test our agent. The first two are the hardest specifications from (Andreas et al. 2017, Toro Icarte et al. 2018). The three additional instructions were defined by aiming for combinations of negated tasks and inclusive/exclusive non-deterministic choices with non-atomic components together with sequential instances. We recall that operator $\cap$ can be defined in terms of sequential composition ; and non-deterministic choice $\cup$.

Table 6.4: Complex instructions used in Table 6.3. The 6 objects included in these instructions belong to the test set ($\mathcal{X}_3$) in the Minecraft-inspired environment, that means every intrusion here is zero-shot.

TTL	Intuitive meaning
$((iron; workbench) \cap wood); toolshed; axe$	Get iron and then use workbench, also get wood. Then use the toolshed. Later use the axe.
$(wood \cap iron); workbench$	Get wood and iron. Then use the workbench
$(grass \sim); grass; (workbench \cup toolshed)$	Get an object different from grass. Then get grass. Later use either the workbench or the toolshed.
$((workbench \sim) \cap (toolshed \sim)); toolshed$	Use an object different from the workbench and use another object different from the toolshed. Then use the toolshed
$((wood; grass) \cup (iron; axe)); workbench; (toolshed \sim)$	Get either wood and then grass or iron and an axe. Then use the workbench. Later use something different from the toolshed.

Importantly, in Table 6.1 we see that the ResNet, while performing poorly with disjunction, shows evidence of systematic learning with negated instructions. This contradicts the results from Hill et al. (2020), where this network struggles with 40 instructions. To check if this is due to be working in a grid-world, we perform additional ablation studies below.

6.2.3 The DRL struggle with negated instructions

This section spots an apparent contradiction between our empirical results and those from the state-of-the-art about generalisation with negation (Hill et al. 2020). It also expands on our motivation to use the reward system of our experiments, which differs from previous research on systematic generalisation, and the BCMs.

We just observed in the previous subsection that the architecture of the CNN plays a key role in the ability of agents to generalise when the number of instructions is limited. Still, Table 6.1 shows that the ResNet achieves good generalisation in both environments learning from 20 negated instructions. This seems to contradict the results from Hill et al. (2020) where the same ResNet struggles to generalise with 40 negated instructions. Nevertheless, we find that this is caused by the particular form that neural networks overfit to the negation operator.

In particular, consider the TTL task *copper* $\sim$, which intuitively means “get something different from copper”. When agents overfit they no longer interpret the negation operator as a separated element, but rather as if the operator was part of the object’s name. Consequently, for an overfitting agent, *copper* $\sim$ is a single word that represents all the objects of the training set except copper. This overfitting phenomena produces policies that, when facing non-training objects, are no longer able to interpret negation correctly, leading to behaviors that differ from what an agent learning systematically is expected to do. We find that this phenomena, first described in Hill et al. (2020), happens only *after* agents overfit to the training set. This means that early stopping is key to correctly evaluate the ability of agents to generalise.

Table 6.5: Results from a ResNet in the Minecraft setting when using a validation set or when not, i.e., the latter are the results from an agent where we did not perform early stopping and overfitted. We see that the different behaviours with positive and negative when the agent is overfitting align with the results from Hill et al. (2020). Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

ResNet ($ \mathcal{X}_1 = 20$)	Not early stopping		Early stopping	
	Reliable	Deceptive	Reliable	Deceptive
Positive instructions	3.08 (1.02)	1.34 (0.65)	3.30 (0.49)	1.10 (0.81)
Negated instructions	1.18(0.46)	1.61(0.72)	3.09 (0.49)	1.26 (0.93)

Ablation experiments. Table 6.5 presents the results of an BCM ablation about the use of a validation set with a ResNet agent trained until convergence in the Minecraft large ($|\mathcal{X}_1| = 20$) training set. We see that there is little difference with positive instructions, but results are completely opposite in the case of negated tasks. Specifically, when applying a validation set and early stopping the agent correctly generalises negation, while they do the opposite from what the specification requires when we do not perform early stopping.

Motivation for our reward system and the BCMS. Performing early stopping is one of the main reason why we use our reward system (detailed in 6.1.2), which differs from what is typically used in this line of research (Bahdanau et al. 2019, Lake 2019, Hill et al. 2020), and required us to perform the BCMS experiment with reliable vs deceptive instructions to assess systematic generalisation. Specifically, previous works reward agents with +1 if they reach the right object and simply end the episode with no rewards if reaching the wrong item. However, note that our agents are not only learning negation instructions but also positive instructions and the disjunction operator (p. and d. instructions for short). When already overfitting to the training set, DRL agents still can improve their success rate with p. and d. instructions, meaning that they still target more often the right object than the wrong one, which differs from what happens with negation. Overfitting with p. and d. is first observed from the agent’s indecisiveness to select a test object, i.e., the agents spends longer and longer time wondering around the environment without reaching any object. This was a strong motivation to introduce the small penalisation per time step. Additionally, the procedural nature of our environments means that for some maps the agent may have learnt that sometimes it is beneficial to go through an undesired object to reach the objective.

Moreover, we observed that *the mainstream metric employed in related literature including Hill et al. (2020) was not capturing effectively the emergence of systematicity even with the negated instructions*. Specifically, previous works were simply measuring how often the agent moved towards the right object and used that alone to measure success. That is, if the agent was choosing the right object 50% of the time, the agent would be discarded as having no generalisation since that performance is the one obtained from random choice. However, we observed that many times agents that were choosing the right object $\sim 50\%$ of the time were only choosing the wrong object $\sim 10\%$, which indicates some generalisation already and a behavior that differs significantly from random. In those cases what happened the remaining times is that the agent did not choose any of the objects before the time limit.

All of this motivates our reward system and our deceptive/reliable BCMs experiments to correctly assess the emergence of systematic generalisation. Consequently, the struggle with negated instructions in previous research seems to be motivated by the various ways deep learning overfits when learning different abstract operators, the lack of validation sets and early stopping, bad performance metrics in previous RL literature.

6.3 Discussion

In this chapter we presented a foundational study that acts as a bridge between the fields of DRL, formal methods, and systematic generalisation. As regards generalisation, we have shown that DRL agents can achieve systematic generalisation of abstract operators – including negation – with just six training instructions. Our results contradicts previous beliefs (Hill et al. 2020) in published research which suggested that DRL is specially prone to overfit abstract operators. A limitation of this work is that all the environments, while visually complex, are regular gridworlds. Nonetheless, this does not lessen our contributions since we demonstrated that the overfitting phenomena observed in previous research also occurs in these settings. Moreover, working in regular worlds allowed us to identify relevant properties within CNN architectures. We remark that in previous research the convolutional encoder is typically a background feature (Bahdanau et al. 2019, Lake 2019), or it is simply stated that most relevant feature is using a deep ResNet (Cobbe et al. 2019, Hill et al. 2020). We demonstrated

that a shallow CNN with random weights can generalise better than a trained ResNet, even when performing early stopping. While it is difficult, if not impossible, to handcraft SA architectures in non-regular worlds, our findings show that object-oriented encoders (Baldassarre & Azizpour 2022) are a promising subject for future work. Our contribution also suggests that previous “disappointing” results in DRL that called into question systematic generalisation in machine learning (Bahdanau et al. 2019, Lake 2019), might have turned differently with carefully selected visual encoders.

As for the literature on formal methods and RL, the main limitation of our work in this chapter is that we use a language that is less expressive than the most recent contributions in the field. Yet this constraint was needed to carry out a careful experimental evaluation of systematic learning with the various types of operators. Given the positive results, more expressive languages are used in subsequent research (Chapter 7). Nevertheless, this work presents the first framework that, leveraging on a learning-oriented syntax and the compositional structure of logical formulae, is capable of executing zero-shot complex specifications in temporal logic while relying on a single network. As anticipated in Section 3.2, this contrasts with dominant methods in the formal methods with RL literature that rely on a new policy network per task (e.g., (Andreas et al. 2017, Icarte et al. 2019, Jothimurugan et al. 2021, Furelos-Blanco et al. 2021)) or a new network per word and symbol (Kuo et al. 2020).

We believe the work in this chapter can help to open new directions of study, as can be seen in Chapter 7 below or in new frameworks inspired from this research (Vaezipoor et al. 2021). Our work also opens the door to more robust reinforcement learning agents that are able to extrapolate their learning to unseen and more critical settings.

Chapter 7

Generalising through Latent-Goal Architectures

That complex instruction the human gave me just requires that I move forward in this timestep. Chapter based on León et al. (2022c)

In Chapters 3 and Chapter 6 we have seen that generalisation in DRL has been traditionally linked to training autonomous agents with large numbers of instructions requiring to build manually those natural language expressions and their corresponding reward functions, which prevents such methods from scaling efficiently (Lake 2019, Vaezipoor et al. 2021). These considerations inspired research on agents learning from formally specified instructions (Wen & Topcu 2016, Alshiekh et al. 2018, Simão et al. 2021) as a substitute for natural language. Earlier contributions in this area rely on the compositional nature of formal languages, often employing multiple policy networks to execute temporal logic (TL) instructions (Andreas et al. 2017, Icarte et al. 2018, Kuo et al. 2020). However, these approaches do not scale well with the number of instructions since policy networks are a computationally costly resource and, consequently, these earlier studies are restricted to relatively small state-spaces environments that require less computation, e.g., non-visual settings. This motivated the work presented in Chapter 6, and continuations like Vaezipoor et al. (2021), that design DRL frameworks capable of generalising to large numbers of out-of-distribution (OOD, i.e., never seen during training)

instructions while relying on a single policy network. These works provide growing evidence that deep learning architectures within DRL agents are key towards the ability of agents to learn formal instructions. Closer to the work in this chapter, Andreas et al. (2018) proposes using natural language as a latent space to improve few-shot learning in image classification, text editing, and reinforcement learning tasks. Through pre-training their model evidenced that creating latent representations from language to structure the learning process can improve generalization and interpretability.

However, network architectures have typically followed a standard configuration, where all the network’s layers except for those encoding the input (e.g., convolutional layers for images, recurrent layers for text) have equal access to both human instructions and environment’s observation. We propose a novel configuration to help agents generalise better by having a task-agnostic representation of the current state concatenated to a latent goal that is formed by processing both observation of the agent and human instruction. As a motivating example, let us consider two scenarios: 1) a robot that is at the centre of an empty room with a red square at the bottom right corner, 2) a robot at the same position in a room that is identical to the one in (1) except that the red square is now green. Intuitively, we can say that, if we give the instruction “go to the red square” in (1), the goal of the robot is the same as if we say “go to the green square” in (2), because the two tasks can be abstracted as “turn to face the bottom right of the room, then move straight”. In a nutshell, computing the human instruction together with the current state of the environment – being at the centre of the room, with the object asked by the human in the bottom right corner – allows to deduce that the optimal policy is the same in both scenarios. Building on that intuition, our **contributions** in this chapter are as follows:

1. We propose a new deep learning configuration (the latent-goal architecture) that helps agents to generalise better when executing multi-task instructions in OOD scenarios.
2. We are the first to evaluate the performance of multiple state-of-the-art (SOTA) neural networks targeting generalisation when following temporal logic instructions.
3. Through ablation studies, we find that networks generating independent pieces of information within their output particularly benefit from being used within latent-goal architectures.
4. We are the first to demonstrate systematic learning of negated atoms in DRL agents with the

classic interpretation of negation, i.e., where not a means that a is false.

The remainder of the chapter is structured as follows: Section 7.1 presents the formal language used to generate instructions and the neuro-symbolic framework that agents use in our experiments. Then, Section 7.2 details the proposed network configuration. Section 7.3 includes the experimental settings, empirical results and ablation studies. Last, Sections 7.4 present the conclusions of our work.

7.1 Learning to solve SATTL instructions

In this section we detail the shared features of our agents. Concretely, Sec. 7.1.1 introduces the formal language we use to procedurally generate instructions, while Sec. 7.1.2 details the neuro-symbolic framework that we use in the experiments of this chapter.

7.1.1 Safety-aware Task Temporal Logic

We investigate agents learning compositionally instructions in temporal logic. Below, we formally define the language from which we generate instructions to be executed by our agents. Specifically, we extend task temporal logic (TTL), the language we introduced in Chapter 6. In that chapter we targeted a simple language where we could assess generalisation in reachability goals such as “eventually reach something different of a sword”. In this chapter, we evaluate agents that handle safety constraints as well, e.g. “walk on soil or stone until you reach a sword”. Thus, we extend TTL to safety-aware task temporal logic (SATTL).

Definition 7.1 (SATTL). *Let AP be a set of propositional atoms. The sets of literals l , atomic tasks α , and temporal formulas T in SATTL are inductively defined as follows:*

$$l ::= +p \mid -p \mid l \vee l$$

$$\alpha ::= lUl$$

$$T ::= \alpha \mid T;T \mid T \cup T$$

Literals l are positive ($+p$) or negative ($-p$) propositional atoms, or disjunctions thereof. Atomic tasks α are obtained by applying the temporal operator *until* (U) to pairs of literals. An atom $\alpha = lUl'$ is read as *it is the case that l until l' holds*. Temporal formulas T are built from atomic tasks by using sequential composition ($;$) and non-deterministic choice ($\cup$). Note that U and $\cup$ are different operators. We use an explicit positive operator ($+$) so that both positive and negative tasks have the same length. This is beneficial for learning negative literals ($-p$) when instructions are given visually, as highlighted in (León et al. 2020). The temporal operators *eventually* $\diamond$ and *always* $\square$ can be respectively defined as $\diamond l \equiv \text{true}Ul$ and $\square l \equiv lU\text{end}$, where *end* is a particular atom true only at the end of the episode. As TTL, SATTL is interpreted over finite traces λ , which in this context refers to finite sequences of states and actions. We denote time steps, i.e., instants, on the trace as $\lambda[j]$, for $0 \leq j < |\lambda|$, where $|\lambda|$ is the length of the trace. While, $\lambda[i, j]$ is the (sub)trace between instants i and j . A *model* is a tuple $\mathcal{N} = \langle \mathcal{M}, L \rangle$, where $\mathcal{M}$ is a POMDP, and $L : S \rightarrow 2^{AP}$ is a labelling of states in S with atoms in AP .

Definition 7.2 (Satisfaction). *Let $\mathcal{N}$ be a model and λ a finite path. We define the satisfaction relation $\models$ for literals l , atomic tasks α , and temporal formulas T on path λ as follows:*

$$\begin{aligned}
(\mathcal{N}, \lambda) \models +p & \quad \text{iff} \quad p \in L(\lambda[0]) \\
(\mathcal{N}, \lambda) \models -p & \quad \text{iff} \quad p \notin L(\lambda[0]) \\
(\mathcal{N}, \lambda) \models l \vee l' & \quad \text{iff} \quad (\mathcal{N}, \lambda) \models l \text{ or } (\mathcal{N}, \lambda) \models l'. \\
(\mathcal{N}, \lambda) \models lUl' & \quad \text{iff} \quad \text{for some } j \leq |\lambda|, (\mathcal{N}, \lambda[j, |\lambda|]) \models l', \text{ and for every } t \in [0, j), \\
& \quad (\mathcal{N}, \lambda[t, j-1]) \models l \\
(\mathcal{N}, \lambda) \models T; T' & \quad \text{iff} \quad \text{for some } 0 \leq j < |\lambda|, (\mathcal{N}, \lambda[0, j]) \models T \text{ and } (\mathcal{N}, \lambda[j+1, |\lambda|]) \models T' \\
(\mathcal{N}, \lambda) \models T \cup T' & \quad \text{iff} \quad (\mathcal{N}, \lambda) \models T \text{ or } (\mathcal{N}, \lambda) \models T'
\end{aligned}$$

Intuitively, by Definition 7.2 an atomic task $\alpha = c_\alpha U g_\alpha$ is satisfied if the “safety” condition c_α remains true until goal g_α is fulfilled. In the context of this work we are not interested in strict safety warranties but in agents trying to reach a goal while aiming to minimise the violation of additional conditions. Thus, we may say our agent has satisfied α even though it has not fulfilled c_α at every time step before g_α . Formally, in those cases we consider traces that are not starting at the beginning of the episode, but from the state after the last violation. The agent is penalised for this behaviour through the reward

function. Examples of tasks we use are $\alpha_1 = \neg \text{grass} U (+\text{axe} \vee +\text{sword})$ “avoid grass until you reach an axe or a sword” and $\alpha_2 = (+\text{soil} \vee +\text{mud}) U \text{axe}$ “move through soil or mud until reaching an axe”. It is not difficult to prove that SATTL is a fragment of the well-known Linear-time Temporal Logic over finite traces (LTL_f) (De Giacomo & Vardi 2013). We provide a translation of SATTL into LTL_f and the corresponding proof of truth preservation in Appendix B.3.

7.1.2 Neuro-symbolic agents

In this chapter we use a neuro-symbolic framework similar to the one presented in Chapter 6 and adapt it to work with SATTL. This section provides details about the main changes, that entails a symbolic and a neural module. Intuitively, given a formal instruction T in SATTL, we again have a symbolic module (SM) that decomposes T into a sequence of atomic tasks α to be solved sequentially by a neural module (NM), which is instantiated as a DRL algorithm.

Symbolic module. Our symbolic module is analogous to the one presented in Chapter 6 with small modifications in the progression subroutine to adapt the module to SATTL. Since this chapter’s scope is introducing novel deep learning network architectures, we refer the reader to Appendix B.1 for further details on the specific changes. Again, we derive our reward function R from a labelling function $\mathcal{L}_I : Z \rightarrow 2^{AP}$. Given a task α , R is defined so that the agent receives a positive reward (+1) if the goal condition becomes true ($\mathcal{L}_I(z_t) = g_\alpha$), a strong penalisation (−1) if the agent is neither satisfying the goal nor the safety condition ($\mathcal{L}_I(z_t) \neq g_\alpha$ or c_α), and a small penalisation (−0.05) if none of the previous is true. The latter penalisation is meant to encourage the agent to reach the goal.

Neural module. The NM consists of a DRL algorithm interacting with the environment to maximise the discounted cumulative reward from R . All our agents use the same A2C algorithm introduced in Chapter 2. To facilitate a fair comparison, all the deep learning architectures use the same visual encoder and output layers for the respective benchmarks. As anticipated, the core of the approach presented in this chapter is the deep learning architecture that we detail below.

7.2 Latent-goal architectures

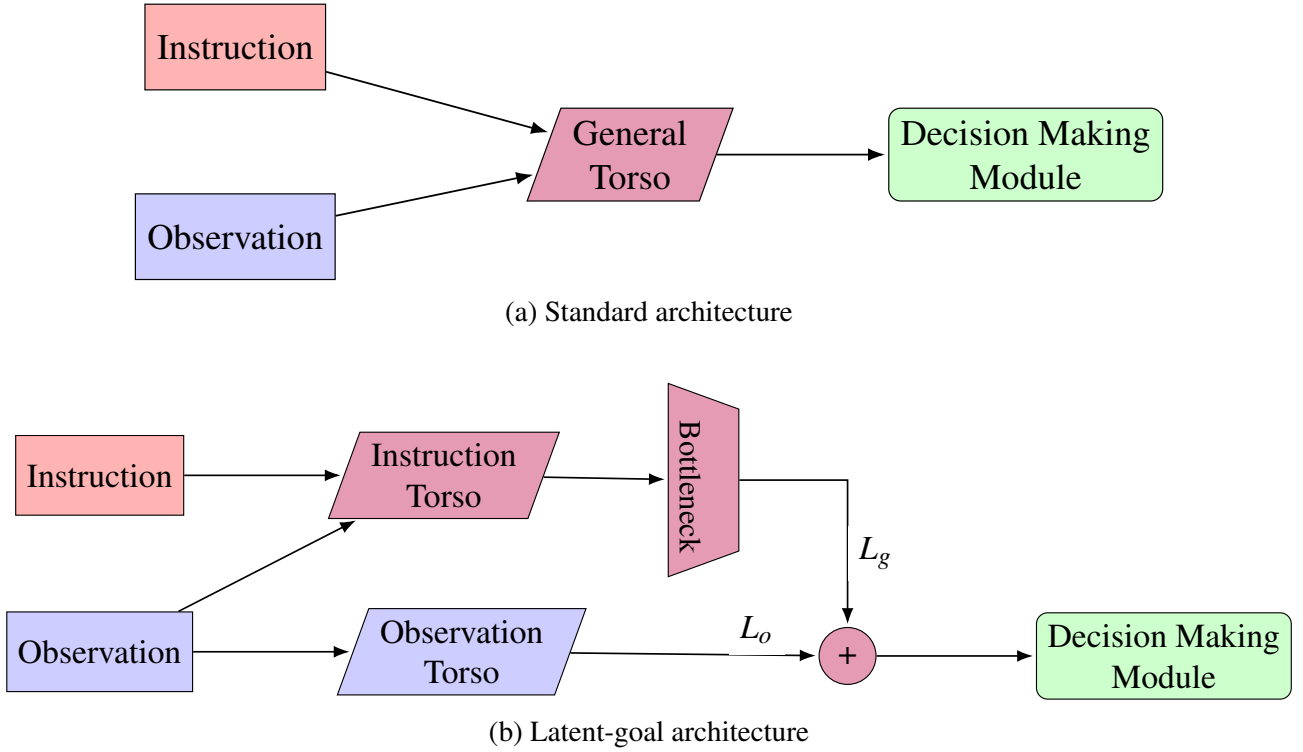


Figure 7.1: Schematic representation of **a)** typical architecture present in previous literature, **b)** *latent-goal* architecture we propose in this chapter. Key design features of our architecture are that the latent-goal (L_g) is the output of an information bottleneck while the latent representation of the observation L_o is task-agnostic.

We presented the language employed to generate instructions (Section 7.1.1) and the neuro-symbolic framework to train our agents (Section 7.1.2). Now we move towards the main focus of this chapter, presenting a new neural network configuration that improves OOD generalisation when executing novel multi-task instructions.

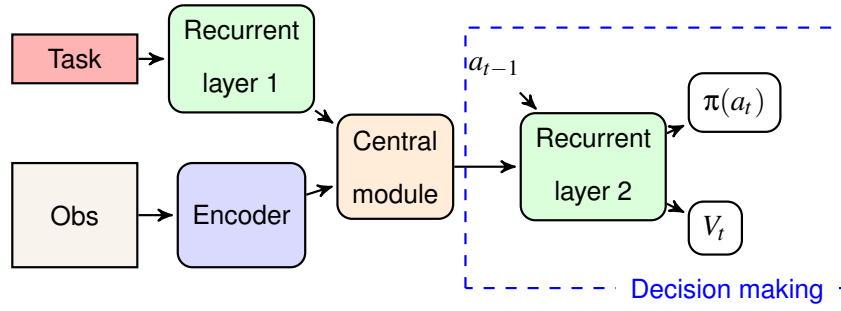
Literature about autonomous agents with generalisation-specialised architectures such as relational deep layers (Shanahan et al. 2020) or modular hierarchical networks (Mittal et al. 2020), commonly follows a standard model architecture set-up where the output of a visual encoder is passed through a central module (CM), which in our context is either a relational layer or an MLP, followed by a recurrent layer. Unfortunately, deep learning agents struggle to generalise instructions compositionally (Lake 2019), even when generalising between task-similar instructions.

Consequently, we propose an architecture configuration that induces agents to compute both human instructions and sensory inputs in a dedicated channel, generating a latent representation of the current

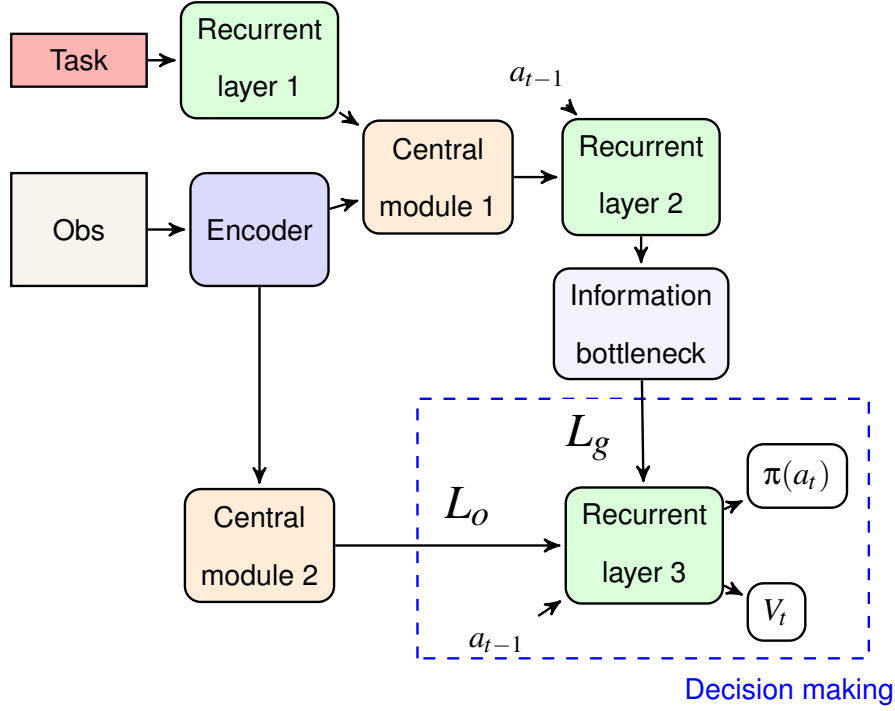
goal. We call this type of deep learning model a *latent-goal architecture*. Figure 7.1 bottom shows our proposed configuration when instructions are given as text; while Figure 7.1 top represents the standard architecture from previous literature for comparison (Hill et al. 2020, Küttler et al. 2020, Vaezipoor et al. 2021). In our configuration, the key feature is that the decision-making layers receive two input streams L_g and L_o . L_g refers to the latent goal, i.e., a representation of the agent’s current goal given the state and the human instructions, and L_o is a vector that contains information about all sensory input processed by the agent except for the instruction. For a latent-goal architecture to generalise effectively there are two central design properties: i) L_o should be task-agnostic, i.e., the layers computing L_o should have no access to what the current task is. ii) L_g should be passed through an information bottleneck, e.g., a low-dimensional fully-connected layer (FC).

Condition i) forces the layers computing L_o to generate representations of the states as generic as possible. By granting no access to the current task, these layers cannot discern which elements provide penalisation or rewards, encapsulating information in a fashion that facilitates generalisation when executing novel instructions. Condition ii) ensures that decision-making layers rely on the generic representation from L_o to extract most of the data from the environment, while the layers processing L_g should focus on providing enough information about the goal.

Hierarchical architectures. The conditions outlined above are flexible enough that a latent goal architecture can be seen as a network with task-agnostic residual connection where there is a bottleneck in the main flow of information. This allows for the latent-goal architecture to be suitable for diverse configurations. Figure 7.2 presents the schema followed by the deep learning networks that we will analyze in the main experiments of this chapter. In this case, the bottleneck is simply a small FC as outlined above. Nevertheless, our configuration can be easily implemented in existing state-of-the-art hierarchical architectures like the bidirectional recurrent independent mechanism (BRIMS) from Mittal et al. (2020). BRIMs are modularised hierarchical networks that hold promise for out-of-distribution generalisation in RL. A particularly interesting feature of BRIMs for our method is that they employ a sparse attention mechanism to communicate between different hierarchies of layers. Specifically, each layer is composed of various modules, and at each time step only the top k modules (according to a key-value attention mechanism) will be able to communicate with modules from dif-



(a) Standard Architecture Design



(b) Architecture that induces Latent Goals

Figure 7.2: Neural network architectures used in the main experiment sections. **Top:** standard architecture from previous literature. The central module can be either fully-connected or relational layers. Outputs $\pi(a_t)$ and V_t refer to the actor and critic respectively (Mnih et al. 2016). **Bottom:** proposed latent-goal architecture.

ferent layers, where k is a hyperparameter. Thus, when using latent goals with BRIMs we can use a task-agnostic residual connection like the one depicted in Figure 7.3.

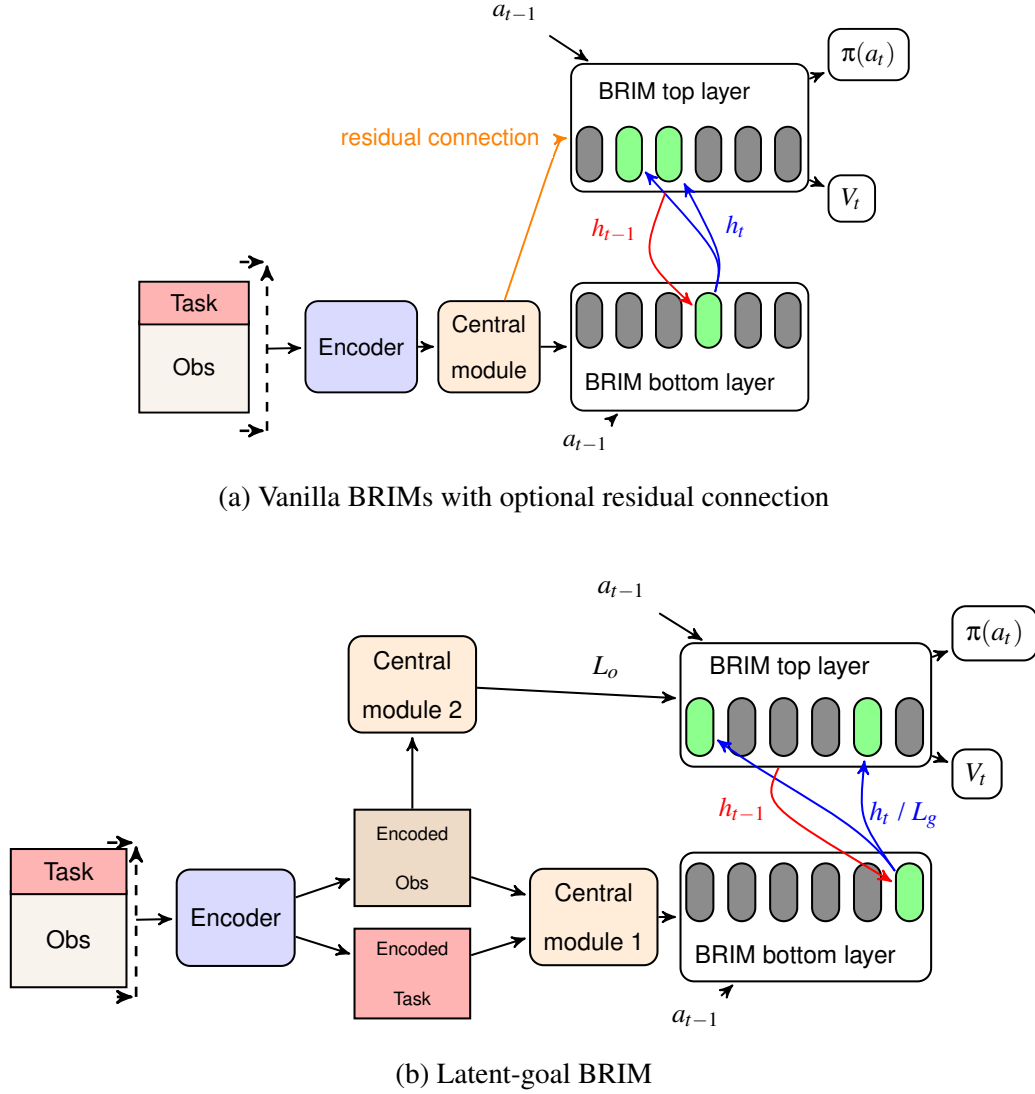
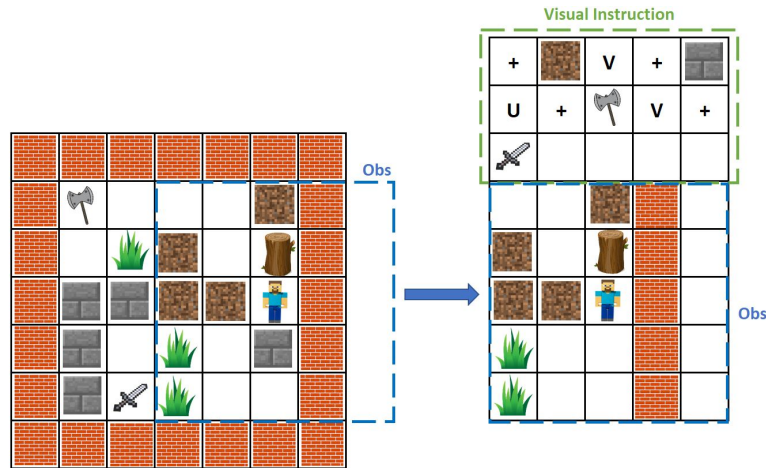


Figure 7.3: **Top:** vanilla BRIM and ResBRIM (without and with the residual connection respectively). Green cells within the BRIM layer refer to activated modules, i.e., those selected by the top-k attention, while gray cells are deactivated. The bottom layer receives as input the output of the central module and the hidden state from the upper layer in the previous time step (h'_{t-1}). The upper layer receives the output from the bottom layer (h_t). Only strong connections (the ones between activated modules) are shown. In the case of ResBRIM the upper layer also receives the output of the Central module. **Bottom:** BRIM^{LG}. The architecture is similar to the ResBRIM but the top layer receives as additional input an embedding from the observation exclusively (L_o). Consequently, the top layer has only access to the current goal through the lower layers of the hierarchy.

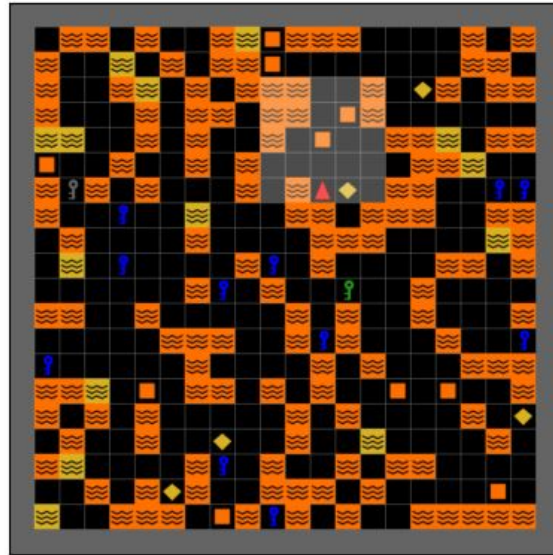
7.3 Experiments

This section starts presenting the experimental settings and continues with the main experimental results, which highlight the benefits of latent-goal architectures. We finish with ablation studies to better understand the inner working of the novel configuration.

7.3.1 Empirical setting



(a) Minecraft train map and observation



- reach orange lava U + reach grey key V + reach forest key

(b) MiniGrid test map

Figure 7.4: **Top:** a 7x7 Minecraft map (left) and its corresponding extended observation (right) with the visual instruction “move through either soil or stone until reaching an axe or a sword”. Tasks are specified at the top of the extended observation by depicting objects and SATTTL operators. Each tile has a resolution of 9x9 pixels. **Bottom:** a 22x22 map in MiniGrid where the agent needs to “avoid orange lava until reaching a gray or dark-green key”. Instructions are given through a text channel and each tile has 8x8x3 pixels. Highlighted cells indicate the field of vision of the agent.

In this chapter we use two benchmarks. One is the Minecraft-inspired environment – Minecraft for short – widely used in the RL-TL literature (Andreas et al. 2017, Vaezipoor et al. 2021) and that we also employed in the previous chapter. Particularly, we use the variant from León et al. (2022a) that works with procedural maps and instructions. In this setting, instructions are visually given as part of

the observation. The second benchmark is MiniGrid (Chevalier-Boisvert et al. 2018), which is also a procedural benchmark where instructions are given as text through a separated channel. Agents are trained in maps of random size $n \times n$, for $n = [7 - 10]$, and evaluated in maps of size $n = 7, 14, 22$. Figure 7.4 illustrates maps from both benchmarks. Instructions are SATTL tasks (see α_1 or α_2 in Section 7.1.1 for examples).

We evaluate generalisation to unseen instructions, thus results under label “Train”, refer to performance with training instructions, while “Test” to OOD instructions. In MiniGrid, where agents require grounding words to objects, OOD specifications are formed with combinations of symbols and objects that were explicitly excluded from training, e.g., if an OOD instructions has the safety condition “avoid orange lava” it means that both color “orange” and shape “lava” were only present in training instructions of the type “reach orange lava”. In Minecraft, where tasks are depicted with objects themselves, OOD instructions refer to tasks with zero-shot, i.e., never seen, objects. Since specifications are human-given, we assume access to the resolution of the visual instructions and select an encoder generating a visual embedding where instructions are independent from the rest the observation. This allows the separation of the task from the rest of the observation, which is beneficial for the latent-goal architectures that generate task-agnostic streams of data. The separation also facilitates a fair comparison between the text and visual instruction domains. Both benchmarks have at least 200K tasks for training and 65K OOD specifications. The rest of this subsection is devoted to provide the details of our empirical setting to facilitate reproducibility.

Minecraft-inspired. In this environment agents have four actions: *Up, Down, Left, Right* to move one square from their current position, according to the given direction. Specifically, we use a configuration where maps are procedurally generated by randomly selecting populations of objects, their placement and agent’s starting position, and where each tile within has a resolution of 9x9 gray (1 channel) pixels. We generate 50 objects split into various sets. Each object is represented by a matrix of 9x9 values. These values were procedurally generated with pseudo-aleatory numbers (we used a fixed seed). The global set of objects is referred to as $\mathcal{X}$, where the total number of objects is $|\mathcal{X}| = 55$. The set is partitioned into pretraining set $|\mathcal{X}_1| = 35$, training set $|\mathcal{X}_2| = 20$ and a test set $|\mathcal{X}_3| = 20$, where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_3$, $\mathcal{X}_2 \subset \mathcal{X}_1$ and $\mathcal{X}_1 \cap \mathcal{X}_3 = \emptyset$.

MiniGrid. In this environment there are three possible actions: *Move Straight*, *Turn Left*, *Turn Right*. Here objects are formed from compositions of colors and shapes. We expand the number of shapes and colors from vanilla MiniGrid to have eleven colors and eight shapes. We refer to the total set of colors and shapes as C and F respectively. We also divide the global set of instructions in four categories as follows:

- Reachability atomic goals, which have the form “ $True\ U + p$ ”.
- Negative reachability goals, e.g., “ $True\ U(-p_1 \vee -p_2)$ ” or “ $True\ U - p$ ”.
- Positive instructions such as “ $+p_1\ U + p_2$ ” or “ $(+p_1 \vee +p_2)\ U(+p_3 \vee +p_4)$ ”.
- Negative conditions, e.g., “ $-p_1\ U(+p_2 \vee +p_3)$ ”.

With atomic reachability goals we use sets C_1 and F_1 while training and C_2 and F_2 for testing, where $|C_1| = 8$, $|F_1| = 6$, $C = C_1 \cup C_2$, $C_1 \cap C_2 = \emptyset$, $F = F_1 \cup F_2$ and $F_1 \cap F_2 = \emptyset$. The remaining categories of instructions use C_3 , F_3 in training and C_4 , F_4 for testing, where $|C_3| = 8$, $|F_3| = 6$, $C = C_3 \cup C_4$, $C_3 \cap C_4 = \emptyset$, $F = F_3 \cup F_4$ and $F_3 \cap F_4 = \emptyset$. Also, $C_2 \subset C_3$, $C_4 \subset C_1$, $F_2 \subset F_3$ and $F_4 \subset F_1$. Intuitively, we use reachability goals to ground some shapes and color for our agents, and the rest of the instructions for the remaining elements. Then we test the agents with the unseen colors and shapes for each category.

Task generation. We procedurally generate tasks as detailed in Algorithm 5. Where task type (α_{type}) refers to one of the four categories of instructions of MiniGrid as detailed above. Note that in Minecraft all task types have the same population of training objects since all the test objects are zero-shot.

Networks and baselines. All the networks follow either a standard or a latent goal (LG) configuration (see Section 7.2). The encoder is a convolutional neural network (CNN) (LeCun et al. 2015) 3-layer convolutional networks in both settings. Particularly, in Minecraft we use a kernel of 3x3 in the first and second layers and 1x1 in the third one. The number of channels in each layer are 8, 16

Algorithm 5 Task generator

```

1: Input: is_test
2: Randomly select task type  $\alpha_{type}$ 
3: if is_test then
4:   Pop  $\leftarrow$  Collection of test objects for  $\alpha_{type}$ 
5: else
6:   Pop  $\leftarrow$  Collection of train objects for  $\alpha_{type}$ 
7: Randomly select number of goals  $n_g$  in range  $[1, 2]$ 
8: if  $\alpha_{type} \neq 1$  then
9:   Defaults number of safety constrains:  $n_c = 0$ 
10: else
11:   Defaults number of safety constrains:  $n_c = 1$ 
12: if  $\alpha_{type} == 3$  then
13:   Randomly select  $n_c$  in range  $[1, 2]$ 
14: Form an instruction  $I$  with  $n_g$  goals and  $n_c$  safety constrains by selecting different objects from
    Pop for every constrain and goal. return  $I$ 

```

and 1 while the strides are 3, 3 and 1 respectively. Note that this encoder has been designed bearing in mind the resolution of the tiles within so that the latent representations (i.e., the outputs of the encoder) of the tasks are independent from the latent representations of the original observation z . This inductive bias is exploited in the latent-goal architectures. In MiniGrid all kernels and strides are of size 2. The number of channels are 16, 32 and 32 respectively. Note that encoders are pretrained and then frozen, we found that doing this improved the performance of all agents. With MiniGrid's text instructions we use a bidirectional recurrent layer of size 32. In multi-layer configurations the other two recurrent layers are LSTMs of size 128. The information bottleneck is a fully connected layer of size 16.

The recurrent layer in the output is either a long short-term memory (LSTM) (Hochreiter & Schmidhuber 1997), that we use in multi-layered architectures, or a bidirectional recurrent independent mechanism (BRIM) in the case of hierarchical models (Mittal et al. 2020). In MiniGrid, where instructions are given as text, we use a bidirectional LSTM (Schuster & Paliwal 1997).

Last, we consider four variants for the central modules within the architectures:

- *Relation Net* (RN) from (Santoro et al. 2017), designed to dispense FC and LSTM layers with relational reasoning.

- *Multi-head Attention Net* (MHA) (Zambaldi et al. 2018), which combines relational connections with key-value attention.
- *PrediNet* (Shanahan et al. 2020), also combining relational and attention operations but specifically designed to form propositional representations of the output.
- *MLP* consisting of a single FC layer, which is meant to act as baseline against the relation-based modules.

For the central modules in the multilayered architectures, we use the best hyperparameters from a previous ablation study in Shanahan et al. (2020). Particularly, in the case of the MLP we use a single fully-connected layer with 640 units. The RN has a central hidden size of 256 units and an output size of 640 units. The MHA employs 32 head with a key size of 16 and value size of 20. Last, the PrediNet has 32 heads, with key and relation sizes of 16. The latent-goal variants use a fully-connected layer of size 16 for the information bottleneck. Regarding BRIMs, we use the hyperparameters recommended for reinforcement learning in Mittal et al. (2020). Specifically, we use two hierarchical BRIM layers, with 6 modules of 50 LSTM units each. Four modules (k) are active at a given time step (top- k mechanism).

Training plots Figure 7.5 shows the training plots of the standard (top), latent-goal (middle) and BRIM (bottom) configurations in Minecraft (left) and MiniGrid (right). In MiniGrid it can be appreciated a general decrease of performance at the 20M mark. This is because we noticed that in this environment, where the navigational features force the agent to further rely on memory, agents struggled when learning with the default configuration of random sizes with $n = [7 - 10]$. To alleviate this, through the first 20 M steps we give a higher chance (70%) to maps of size 7, to help agents to learn from the instructions before navigating through larger maps. Note that good training performance, as shown in these training plots, does not necessarily mean good OOD performance, as we will see below.

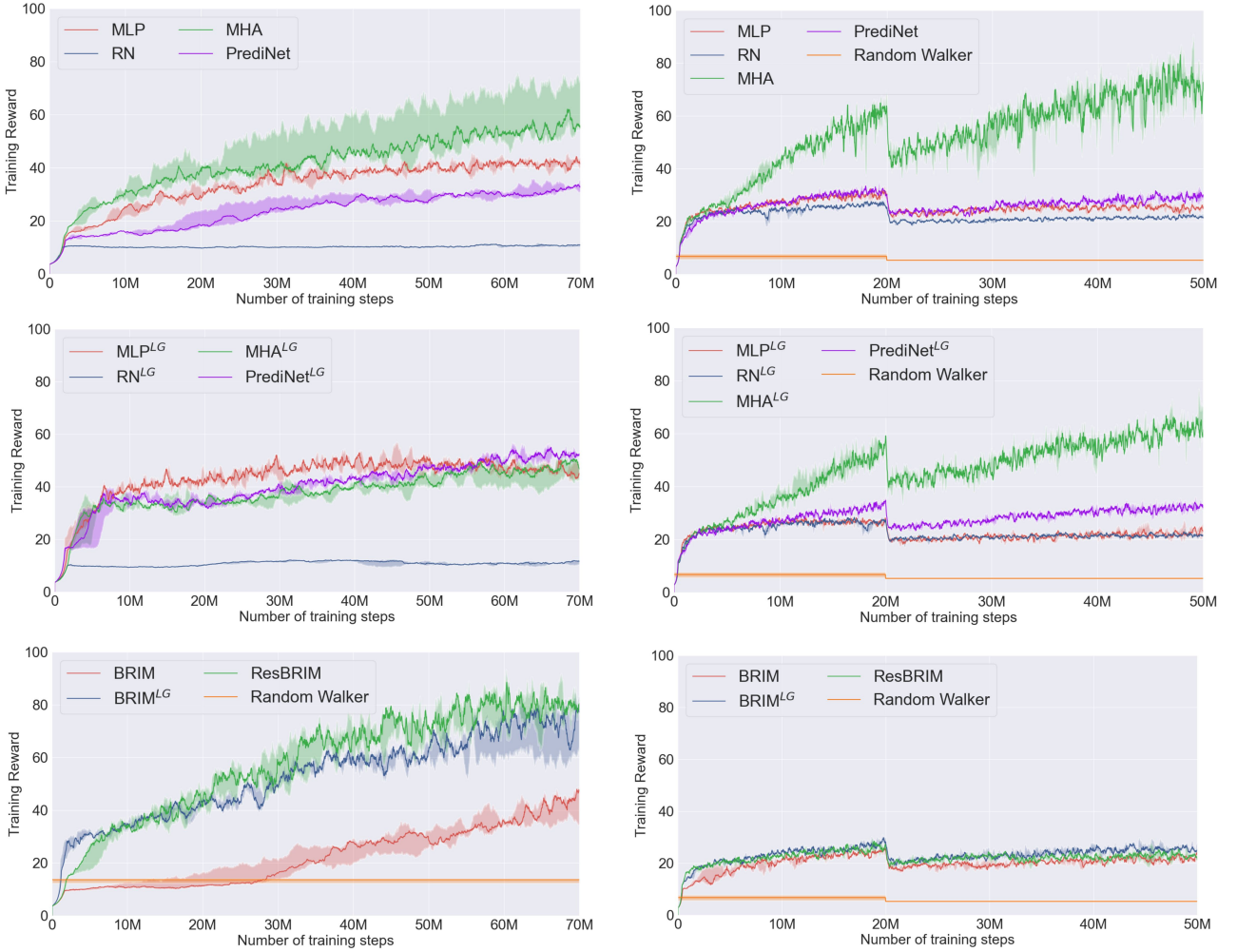


Figure 7.5: Training plots of the networks studied in Section 7.3.2. Continuous lines correspond to the 50th percentiles while the shadowed areas are the 25th and 75th percentiles. Note that good train performances do not yield to similarly good test results as detailed in the main text. **Left:** Minecraft training plots. **Right:** MiniGrid plots. Note that the general performance drop at 20M in this setting is motivated by the higher chance of generating larger training maps from that point on.

7.3.2 Experiment Results

Below we dive into the main empirical results with latent-goal architectures.

Results with multi-layer architectures. Figure 7.6 shows the train performance (top) and zero-shot performance (bottom) of the different multi-layer networks (see Figure 7.7 for the reference of a random walker). Results that are labelled as CM networks (e.g., “RN”) refer to standard configurations, whose CM uses the network of the label. Correspondingly, results with the LG superscript (e.g., RN^{LG}) refer to latent-goal configurations. Overall, we see that our proposed configuration

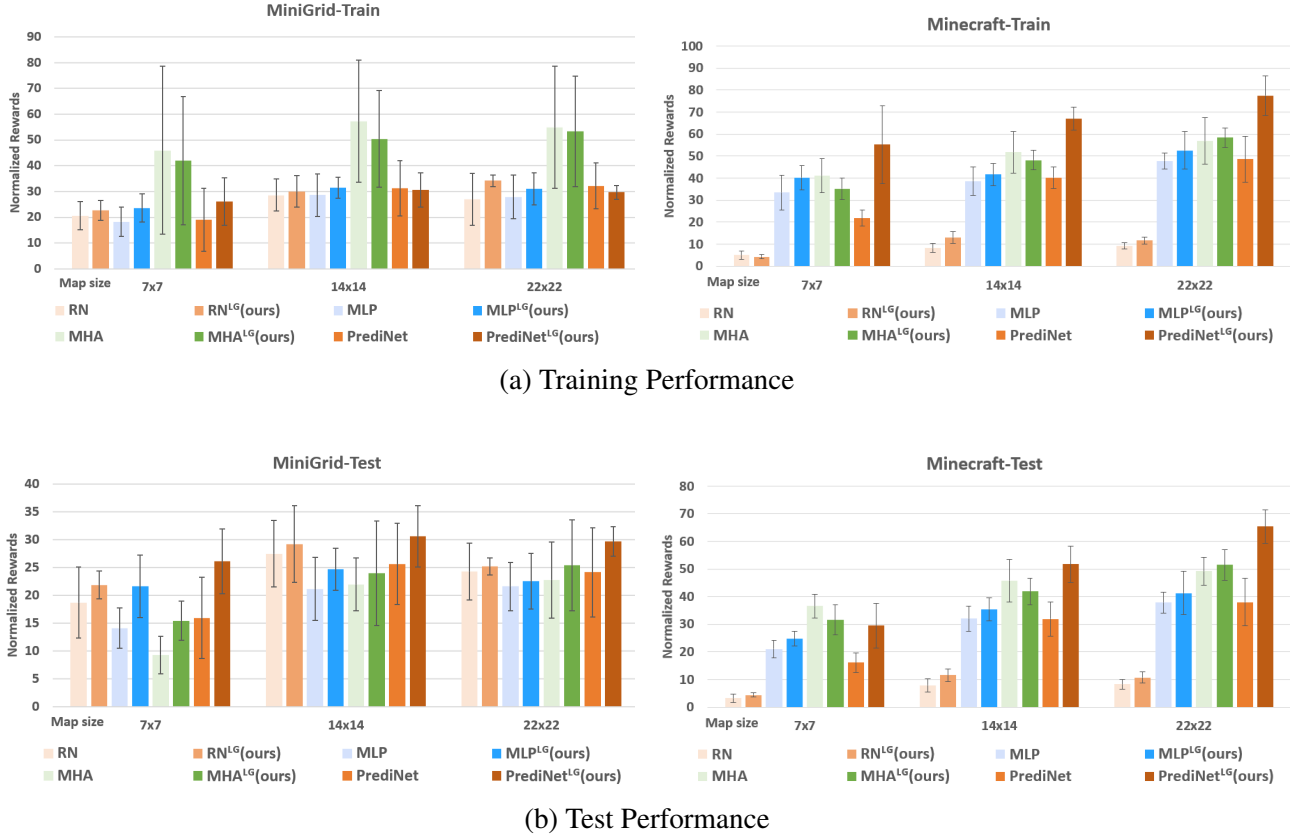


Figure 7.6: Results with training (top) and out-of-distribution (OOD) instructions in 500 maps (per size) of different dimensions. Sizes 14 and 22 are OOD. MC refers Minecraft whereas MG to Mini-Grid. Results show the average reward and standard deviation from 10 independent runs. Values are normalised so that 100 refers to the highest performance achieved by the best run globally in maps of the given size and benchmark.

improves the performance of all the networks in most zero-shot scenarios. Moreover, a latent-goal configuration is always the best performer in the hardest test settings ($n = 22$), where the best latent-goal model achieves 33% and 22% better performance compared to the best standard architectures in Minecraft and MiniGrid, respectively. Training results and tables with the specific values are given in Appendix B.2. We also observe that the PrediNet is the network that benefits the most from the new configuration (up to 74%) and PrediNet^{LG} becomes clearly the best architecture. Noticeably, the two most different results with latent-goal architectures come from the networks that combine relational and attention mechanisms (MHA and PrediNet). This point is further examined in the ablation below.

Results with hierarchical architectures. Results also demonstrate that latent goals can improve the performance of hierarchical architectures. In the next experiments we fix the PrediNet as the default network for the different CMs – note that the PrediNet^{LG} achieved the best results in the

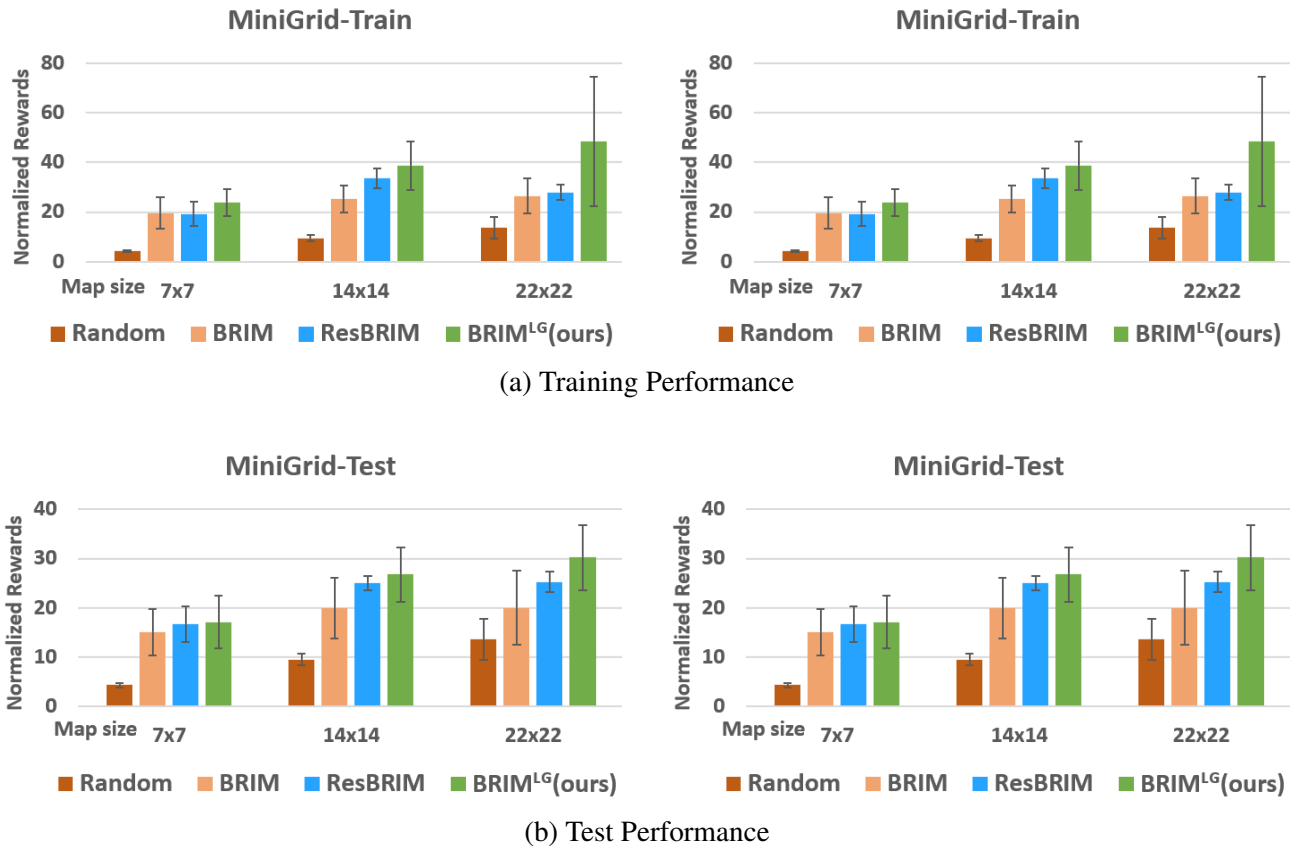
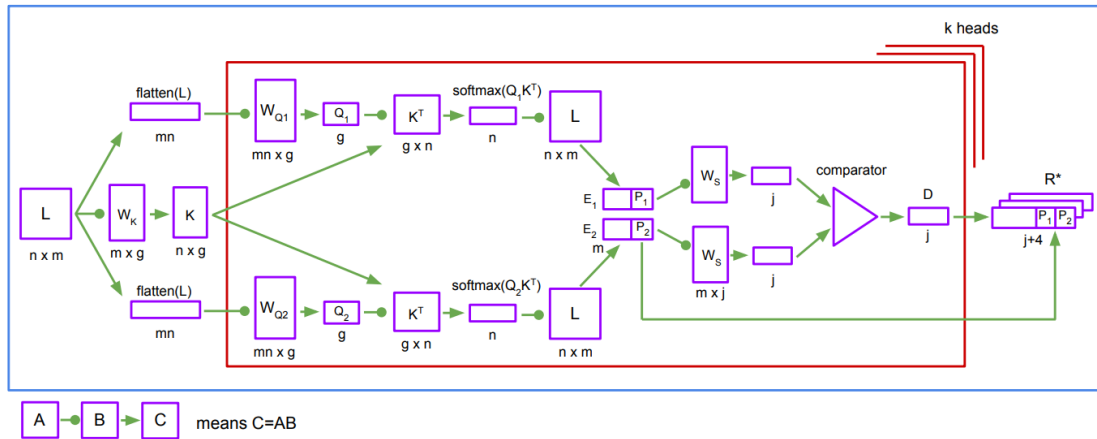


Figure 7.7: Results of 5 independent runs of vanilla BRIM (BRIM), BRIM with residual connections (ResBRIM), and with LGs (BRIM^{LG}).



previous evaluation – and replaced the last recurrent layer of the architectures with a BRIM. BRIMs are modularised hierarchical networks that hold promise for OOD generalisation in reinforcement learning. Since in BRIMs only the top k modules can forward information according to a key-value attention mechanism, we remove the small fully-connected layer that acted as information-bottleneck in the previous networks. We do not perform any hyperparameter optimization with BRIMs and use

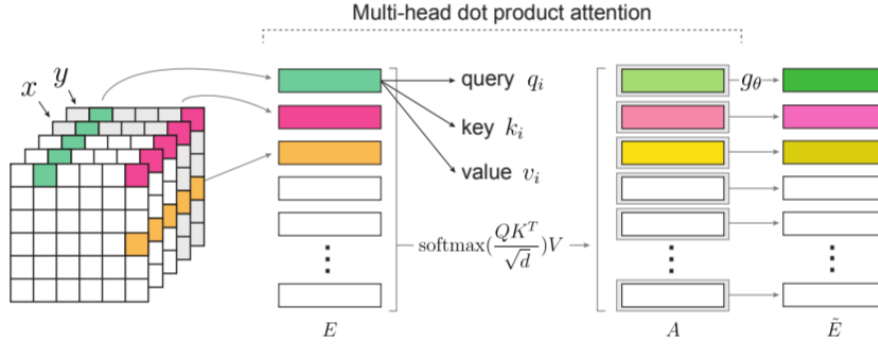


Figure 7.9: The MHA from Zambaldi et al. (2018).

directly the same configuration from the RL experiments in Mittal et al. (2020). Figure 7.7 contrasts the performance when using a vanilla BRIM layer (BRIM) against using a residual connection (He et al. 2016) (ResBRIM) or latent goals (BRIM^{LG}). We see that inducing the latent goals improves the performance of the hierarchical network in all settings. Hence, we see that deep hierarchical architectures can also capitalise on specialised hierarchy levels where bottom layers produce latent representations of the goal while the higher levels receive task-agnostic information of the state.

7.3.3 Ablations

Contrasting MHA and PrediNet with their latent-goal counterparts, we see that the particular differences of the PrediNet with respect to the MHA internal functioning benefits the proposed latent-goal architecture. After a closer inspection, we see that the most significant difference comes from a channeling of the input within the PrediNet (Figure 7.8). This network encourages a kind of the semantic separation of the representations that it learns, making an element-wise subtraction of the extracted features, an operation not found in the MHA (Figure 7.9). Additionally, we see that the PrediNet modifies its output to explicitly represent positions of the pair of features selected by each of its heads, also not present in the MHA. Figure 7.10 illustrates the impact of individually removing these distinct features within the PrediNet and whether the PrediNet is beneficial in both CMs or it is better to use an MHA in any of the modules. We find that the element-wise subtraction is the key feature within the PrediNet since its removal reduces the performance of PrediNet^{LG} to that of an MHA^{LG}. None of the networks using an MHA in one of the CMs outperform having a PrediNet in both.

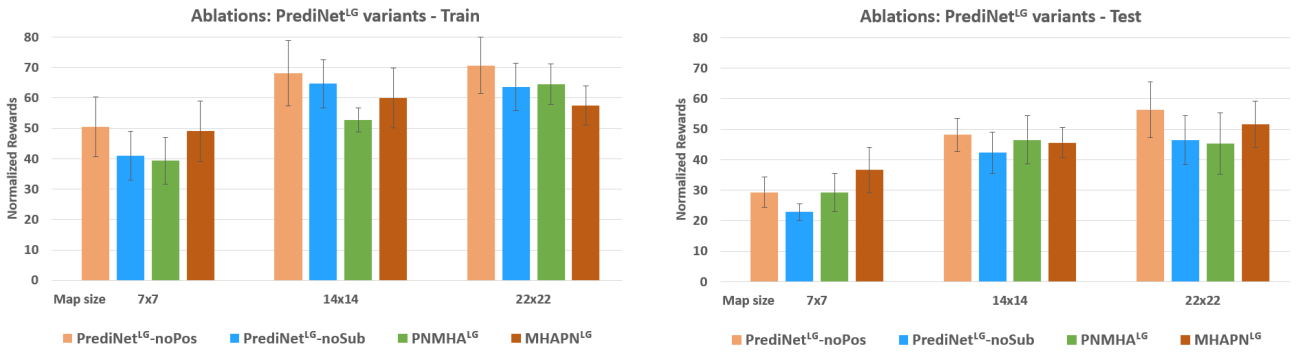


Figure 7.10: Ablation studies in Minecraft (5 i.r. per variant). PrediNet^{LG}_{noSub} and PrediNet^{LG}_{noPos} are variants of PrediNet^{LG} without the element-wise subtraction and the feature coordinates respectively. PNMHA^{LG} uses an MHA in CM2 while MHAPN^{LG} uses an MHA in CM1.

Regarding the outlayer of MHA in MC-14x14, we believe that receiving instructions not preprocessed by a recurrent layer is preventing the MHA to overfit as much as it does in MiniGrid, thus achieving better performance in maps of close-to-distribution size. Still, note that MHA^{LG} outperforms MHA in three out of four OOD results. As for the reason why the standard PrediNet does not outperform the MHA in the same line that PrediNet^{LG} outperforms MHA^{LG}, from Shanahan et al. (2020) we note that at the core of the PrediNet information is organised into small pieces that are processed in parallel channels limiting the ways these pieces can interact. This pressures the network to learn representations where each separate piece of information has independent meaning and utility. The result is a representation whose component parts are amenable to recombination. Such feature is highly beneficial for generalisation, but supposes that the following recurrent layer needs to combine those pieces while producing a meaningful output for both actor and critic. As highlighted in Santoro et al. (2017), recurrent layers struggle when relating independent pieces of information. The latent-goal configuration alleviates this problem by decoupling the problem of goal abstraction and state representation into separated channels. Consequently, a latent-goal architecture helps to exploit the full potential of the highly re-usable outputs produced by PrediNets. Thus, if we substitute the element-wise subtraction with a FC layer (as we do with PrediNet^{LG}_{noSub}), the output is no longer generating independent pieces of information from the parallel channels, which aligns the representations obtained from the PrediNet with the ones generated by an MHA, that employs a single channel of information.

Last, Figure 7.11 provides the results of ablation studies regarding conditions (i) and (ii) - task-agnostic main stream of information and sparse goal information, respectively- from Section 7.2. We

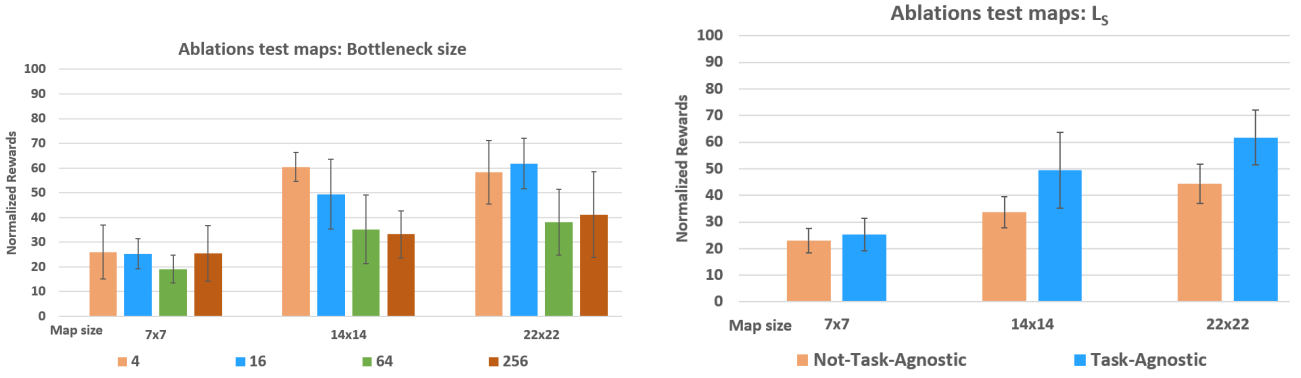


Figure 7.11: Ablation studies with PrediNet^{LG} in Minecraft (5 i.r. per variant). **Left:** Impact of different bottleneck sizes. **Right:** Impact of having L_s task-agnostic or not.

Table 7.1: Study on compositional learning with zero-shot objects and instructions. Results show the performance obtained by each network in 200 test maps when rewards are given according to the “real goal”, while the symbolic module provides the “given instruction” to the neural module. An agent learning compositionally should have 1st row > 2nd row > 3rd row for the values within its column. Also, values lower than random are only acceptable in the third row (deceptive instruction). The mean performance of a random walker is 12.5 independently of the “given instruction”.

Real goal: $-cU + p$								
Given instruction	MLP	MLP ^{LG}	RN	RN ^{LG}	MHA	MHA ^{LG}	PrediNet	PrediNet ^{LG}
$-cU + p$	44.6(10.5)	70.6(11.9)	14(1.1)	13.9 (1)	55.8(6.1)	43.8(7.3)	17.5(1.2)	43.6(9.3)
true $U + p$	20.3(5.7)	18.0(4.6)	13.3(0.9)	13.9(1.4)	16.9(3.0)	19.9 (6.3)	19.9(3.4)	17.5(3.8)
$+cU + p$	8.7(1.6)	11.0(2.6)	7.9(2.2)	5.6(1.2)	10.8(2.6)	15.0(3.6)	8.2(1.8)	8.4(1.2)

observe that, for larger bottleneck sizes, generalisation performance drops significantly as the agent no longer relies on the generic representations from L_s since biased information from L_g is no longer constrained. That is, larger bottlenecks do not limit the information passed through L_g biasing the agent towards the objects used in the training tasks. We see a similar degradation in performance when providing information about the task through L_s , confirming that the benefits are drawn on the generic representations from an input that cannot tell on its own which objects are beneficial or dangerous.

7.3.4 Systematic Learning

Moreover, we perform control experiments in Minecraft to verify that our agents are learning systematically from the training instructions, which as discussed in the previous chapter, earlier works (Hill et al. 2020, León et al. 2022a) had found particularly challenging. Specifically, we assess whether

our agents are learning systematically SATTL instructions and check if they are really following the given safety constraints when applied to OOD objects or if they are just learning to reach the goal g_α . We focus our study on their performance with negated constraints, since as discussed in previous chapters is a challenging type of instructions Hill et al. (2020), León et al. (2022a). Systematic learning (also known as compositional learning Lake (2019)) refers to the ability of understanding and producing novel utterances by combining already known primitives Chomsky & Lightfoot (2002). In our context, an agent that is learning systematically should be able to solve the instruction $-cU + p$ if the primitives c and p are known and it already learnt to solve $-c'U + p'$.

Table 7.1 shows the results from a control experiment where we track the performance of the agents according to tasks of the form $-cU + p$ when giving reliable, partially occluded or deceptive instructions. Specifically, every row shows the test performance of the multi-layered networks trained in Section 7.3 when providing rewards according to the instruction $-cU + p$, which intuitively means “avoid c until reaching p ”. The first row shows the performance of the agents when the SM provides *reliable instructions* to the NM, i.e., the instruction provided is the one used to generate rewards. The second row shows the performance under the same setting, but where the extended observation provided to the NM is a reachability goal ($\text{true}U + p$). Intuitively, for the second row the agents are given the right goal p but not the safety constraint, i.e., partially occluding information of the real task. The third row shows the performance when the SM gives deceptive information about the safety constrain ($+cU + p$), i.e., the SM is “telling” the NM to go through the objects that actually should be avoided. Agents learning systematically should show worse performance with partially occluded instructions than with the reliable ones, but still better than a random walker. Additionally, the worst performance should come when provided deceptive instructions.

From Table 7.1 we see that all the variants using the MHA and MLP modules follow the rule $1^{st} \text{ row} > 2^{nd} \text{ row} > \text{random} > 3^{rd}$ (systematic rule, or s.r. for short). Notably, the best performance comes from a variant that does not use relational networks nor attention (MLP^{LG}). Still, this does not imply that MLPs are better suited for negation since agents were trained in a much wider variety of tasks and the MHA, MHA^{LG} and $\text{PrediNet}^{\text{LG}}$ outperformed the MLP^{LG} in the general evaluation test. In the case of the RN, the s.r. is satisfied but the general low performance and the small difference between the

results of the first and the second rows (14 and 13.3 respectively) indicates weak generalisation. This is worse with the M-RN and the PrediNet having both equal or better performance with partially occluded instructions than when receiving the real task as input. Such results imply that these networks have not correctly learnt to generalise negated instructions. This is not the case with the PrediNet^{LG}, whose results correctly follow the s.r.. In addition, the worse performance of the PrediNet^{LG} than the MHA with partially occluded and deceptive instructions suggests that the PrediNet^{LG} advantage over the MHA in maps of OOD size (see Table B.1 comes from a better ability of the former to execute safety constraints when compared with the later. Note that the larger the map the harder it is to find p ; thus the bigger chances of accumulating penalisations due to "violations" of the safety condition.

Discussion To the best of our knowledge, the only two works that include some empirical evidence of emergent systematic learning with negation, i.e., achieving a performance 50% better than chance are Hill et al. (2020), León et al. (2022a). However, in those works negated instructions are interpreted as "something different from", e.g., "not p " intuitively meant "get something different from p ". Instead, here we have an interpretation of negated atoms more aligned with propositional and temporal logic, and also to natural language, where "not p " intuitively means that p must be false. Hence, this chapter includes the first work to show that DRL agents are capable of learning the abstract operator of negation in its classical interpretation and successfully apply it to new utterances.

7.4 Discussion

By relying on the assumption that human instructions can be separated from the sensory input, we have presented *latent-goal architectures*, a novel deep learning configuration that induces situated agents to generate latent representations of their current goal. Particularly, we have seen that deep learning can draw on configurations where output layers take decisions based on both task-agnostic representations of the environment's state and the outputs of a channel that processes observations and human instructions together, we call those outputs *latent goals*. We have provided evidence of the strong potential this configuration has through its good generalisation results. Latent-goal architectures were tested in OOD visual grid-worlds with five different network configurations that

incorporated state-of-the-art mechanisms such as attention, bidirectional hierarchy, modularity and relational biases. We demonstrated that all the architectures benefited from using latent-goals without any hyperparameter tuning. In this chapter, we also proved for the first time the emergence systematic generalisation with the classical interpretation of negation in a DRL agent, and demonstrated how latent goals abilitate the emergence of systematic learning for state-of-the-art neural network architectures.

Looking ahead, literature (Qiu et al. 2024) on formal methods and Reinforcement Learning (RL) has already advanced our Symbolic Module through more universally applicable methods. Nevertheless, our principal contribution, the *latent goal architectures*, diverges from the reliance on formal languages. This distinction allows our architecture to be beneficial also for the increasingly popular trend of utilising natural language instructions with grounded agents. Thus, a promising continuation of our work will be combining the latent-goal architectures with large networks such as transformers (Parisotto et al. 2020) or the structured space models (Gu & Dao 2023) that have recently become dominant in this domain.

Part III

Safety-aware Policy Exploration

Chapter 8

Discovering Creative Safety-Aware Policies with DUPLEX

I won't master every opponent with the same strategy. Chapter based on G. León et al. (2024).

Part II of this thesis presented novel solutions for better generalisation in deep reinforcement learning (DRL). However, robustness and effective generalisation is also deeply correlated to good exploration (Jiang et al. 2024). As we highlighted in Chapter 3, exploration is particularly brittle when satisfying multiple constraints in RL. Therefore, this chapter is devoted to design AI solutions that thrive to explore in complex environments under constrained objectives. Particularly, environments where RL agents are compelled to exhibit flexible and diverse behaviors to robustly adapt to different scenarios and interact with other acting entities (Vinyals et al. 2019, Zahavy et al. 2024). In Chapter 3 we highlighted the growing community researching methodologies to train agents that, unlike conventional RL (Haarnoja et al. 2018, Espeholt et al. 2018), are capable of solving tasks with a diverse set of near-optimal strategies (Zahavy et al. 2023, Kumar et al. 2020, Cully et al. 2015). Such methodologies are explicitly crafted to enhance exploration of the state-action space, equipping agents with the capability to discover diverse solutions for a given task distribution. However, existing approaches either struggle to generalise to unseen tasks (Zahavy et al. 2023, Kumar et al. 2020) or require evolution as the optimisation method – which can be computationally expensive (Cully et al. 2015, Flageat

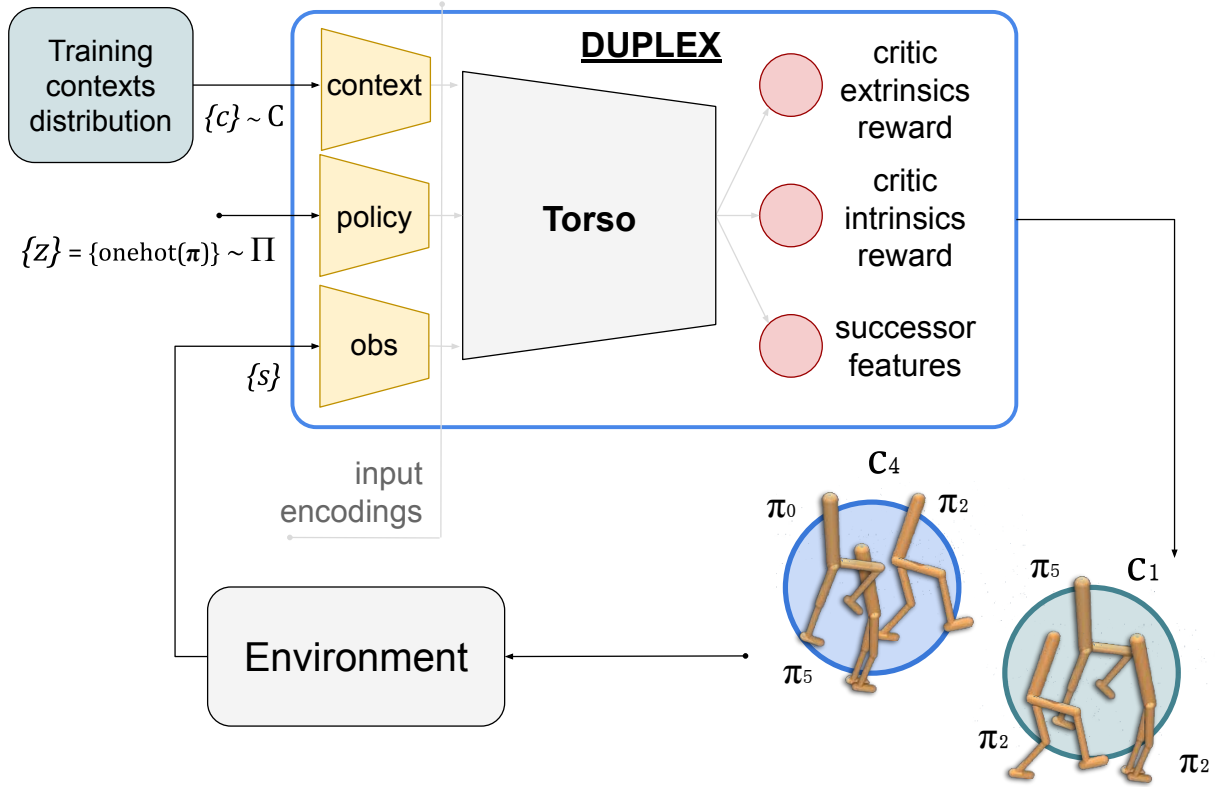


Figure 8.1: DUPLEX data flow training a multi-policy agent over a set of tasks. At each iteration of DUPLEX we provide three inputs to give context about the current operational dynamics and task requirements c ; an encoding of the current policy being used z and the current state of the environment s . Our agent then is trained to output three quantities: critic estimates for the intrinsic and extrinsic rewards and successor features to drive diverse behavior discovery. Finally, the algorithm samples policies in Π uniformly and rolls them out to collect more experience.

et al. 2023). In this chapter we tackle such shortcomings in diversity learning methods, by leveraging universal-estimators to provide better generalisation with unseen tasks (Schaul et al. 2015, Borsa et al. 2019, Carvalho et al. 2023), enabling us to use reward maximisation as our optimisation objective.

We introduce DUPLEX, a framework that learns diverse universal successor features estimators to boost learning of diverse policies. DUPLEX builds on top of the strengths of DOMiNO (Zahavy et al. 2023) for diversity learning in RL and universal estimators (Schaul et al. 2015, Borsa et al. 2019) to present a novel methodology that improves both quality and diversity while overcoming existing limitations. DUPLEX is an iterative algorithm that trains an agent to optimise a set of diverse policies over different environments. Consider an astronaut robot tasked with exploring planets with different environmental conditions and task requirements. DUPLEX is designed to provide a sense of behavioral diversity to the agent, promoting adaptability and robustness even in conditions beyond

its training data – encountering unfamiliar gravity forces, terrain types, air viscosity, and task specifications, such as maintaining specific end-effector positions or ensuring battery levels stay above a certain threshold.

To achieve such a level of performance and representation of the environment, we adopt the *contextual* Markov Decision Process (contextual MDP) (Hallak, Di Castro & Mannor 2015) framework where different episodes correspond to different contexts, i.e. different environments which, nevertheless, share structure. As shown in Figure 8.1, DUPLEX is designed to receive three inputs: the encoding representations of the context, policy, and state. And, with the critic, the output is split into different heads to support the estimation of the extrinsic and intrinsic rewards, and successor features (SFs) (Barreto et al. 2017). We refer to the reward coming from the environment as the extrinsic reward, while to the signal computed to promote diversity as the intrinsic reward. The set of policies is then trained to maximise both the distance in their successor features, guided by intrinsic rewards, and the task performance, given by the extrinsic rewards. We rely on SFs to measure distances as they, by definition, represent the state features that a given policy is expected to experience. Thus, SFs represent a intuitive way to formalise policy diversity. Such an architecture allows us to learn an agent which can simultaneously train a set of policies to operate under a given task distribution while estimating and improving on behavior diversity.

Our main contribution in this chapter is DUPLEX which, to the best of our knowledge, is the first RL framework formalising an approach to effectively train an agent to find a diverse set of near-optimal policies, while able to generalise within and out-of training distribution (OOD). Moreover, we contribute to (i) the state-of-the-art of universal estimators, with a more accurate estimation of SFs that boosts stability and performance, and to (ii) diversity learning in RL, by showing that we better preserve the diversity vs. near-optimality trade-off while providing a more stable solution when SFs are effectively estimated. As evidenced in the experiments, DUPLEX is more robust across different domains and more successful in complex realistic simulators than the state of the art, which cannot manage to learn diverse competitive policies. We evaluate DUPLEX on two multi-task and changing dynamics versions of Mujoco environments (Walker2D, Ant) (Todorov et al. 2012) and on the real-time, physically realistic, car-racing simulator GranTurismoTM 7 (GT) (Wurman et al. 2022). We also

conduct an ablation study to evaluate the different components of our approach and provide further evidence of the advantages of the main features in DUPLEX. Videos of our agent can be found at <https://sites.google.com/view/bgl-phd-duplex/home>.

8.1 Notation

Here we briefly introduce the main DUPLEX building blocks and the notation that we adopt.

Multitask RL. We consider training a multi-policy RL agent that solves a contextual MDP (Halak, Castro & Mannor 2015) that we introduced in Chapter 2. Contextual MDPs are presented as a tuple $\mathcal{M} = \{S, \mathcal{A}, P, R, \gamma, \mu_C, \mu_S\}$, where $S, \mathcal{A}, R$ and P are defined as in an MDP, μ_S is the initial state distribution conditioned on the context, and μ_C is the context distribution. Note that we use the context $c = \{u, w\}$ to summarise both information about the particular dynamics u of that environment (e.g., gravity) and information about the task w in the current episode (e.g., height the agent needs to walk beneath). Every episode starts with a context $c \sim \mu_C$ and an initial state $s_0 \sim \mu_S(\cdot | c)$. Then, at each time step t , the agent selects an action a_t according to its policy $\pi(\cdot | s_t, c)$, receives a reward $r_t \sim R(s_t, a_t, c)$ and transitions to the next state $s_{t+1} \sim P(\cdot | s_t, a_t, c)$. Policies are characterised by their state-action occupancy, i.e., how often a policy samples a state-action pair. We consider two state-action occupancy metrics: $d_{\pi_c}^{\text{avg}}(s, a) = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} \sum_{t=1}^T \mathbb{P}_{\pi_c}(s_t = s) \pi(s, a, c)$ for the average occupancy metric and $d_{\pi_c}^{\gamma}(s, a) = (1 - \gamma) \mathbb{E} \sum_{t=1}^{\infty} \gamma^t \mathbb{P}_{\pi_c}(s_t = s) \pi(s, a, c)$ for the discounted case – where $\mathbb{P}$ is the probability measure of states at t induced by π in c . Our objective, akin to canonical RL, is finding policies that maximise the expected return over the sum of contexts $\max_{\pi} Q^{\pi}(s, a) \equiv \max_{d_{\pi_c} \in \mathcal{K}} \sum_{s, a, c} r(s, a, c) d_{\pi_c}(s, a)$, where $\mathcal{K}$ is the set of admissible distributions (Zahavy et al. 2021).

Universal estimators. When training RL agents to solve multiple tasks, we follow the common approach of decomposing the input of the neural network to facilitate transfer learning between tasks and policies. Methods that are relevant for our work include “Universal Value Function Approximators” (UVFA) (Schaul et al. 2015), which add a task-descriptor vector w as input to a value function approximator parameterised by θ , $V_{\theta}(s, a, w)$. If V_{θ} is smooth with respect to w , then V_{θ} is expected

to generalise across tasks within the training task space. Akin to previous works (Barreto et al. 2017, Zahavy et al. 2023), we assume that every state-action pair is correlated with observable *features* known as “cumulants” $\phi(s, a, c) \in \mathbb{R}^d$. These cumulants can either be given through relevant properties within the state¹ or be learned. In a similar fashion to value functions, we can define *expected features* $\Psi_{\pi_c}(s, a) = \mathbb{E}_{s', a' \sim d_{\pi_c}(s, a)} \phi(s', a', c) \in \mathbb{R}^d$, which we will refer to as average (Ψ^{avg}) or discounted (Ψ^γ) expected features when using $d_{\pi_c}^{\text{avg}}$ and $d_{\pi_c}^\gamma$ respectively. In the discounted case, Ψ^γ are also known as “successor features” (SFs). This latter formalisation is key to our work since we explore a more general formalisation of SFs known as USFA (Borsa et al. 2019) which combines UVFA-like task decomposition with the SFs, and exploits generalised policy improvement (GPI) Barreto et al. (2017). Importantly, GPI guarantees that given a set of policies, we can obtain a new policy that is as good or better than the best policy on the set by taking the action a_t^* at every step t , where $*$ marks the action taken by the policy with the highest expected value of the set in the current s_t . USFA disentangles both tasks and policies by giving as input a vector z that is a representation of the current policy $z = e(\pi)$, where e is an encoding function – in our approach e encodes policies as one-hot vectors z . Such an encoding is further motivated by the work in Borsa et al. (2019) that finds that disentangling inputs leads to more general value functions $V_\theta(s, a, w, z) = \Psi_\theta(s, a, z)^\top w$ facilitating transfer across both tasks and policies.

8.2 Learning Near-Optimal Diverse Behaviors in Multi-Goal Continuous Settings

DUPLEX is a framework designed to provide diverse solutions to complex tasks by simultaneously maximising both the true reward and dissimilarity within a set of policies Π_c across the different environments of a contextual MDP. We first provide the definition of diversity which is key to frame our research objective:

Definition 8.1 (Diversity). *Diversity(Π) is a metric of dissimilarity among policies in a set Π with a common goal. Formally, if Ψ_{π_i} and Ψ_{π_j} are a function of state-occupancy of relevant features of*

¹For example, in a car racing scenario features could correspond to relevant information for the car such as the course velocity or if a car has crashed into another

any two policies in Π , then their dissimilarity is given by $\|\psi_{\pi_i} - \psi_{\pi_j}\|$. A non-zero value of this norm indicates dissimilarity, with larger values indicating greater divergence between the policies. Mathematically, diversity is then defined as the sum of min L-2 dissimilarities norms in Π :

$$\text{Diversity}(\Pi) = \frac{1}{2 \text{size}(\Pi)} \cdot \sum_{\substack{\forall \pi_i, \pi_j \in \Pi, \\ i \neq j}} \min \|\psi_{\pi_i} - \psi_{\pi_j}\|_2^2$$

Intuitively, we define diversity as a measure of state-action occupancy. According to such definition we formulate our learning problem and the major contributions that DUPLEX is carrying with. Note that Definition 8.1 serves as a general definition only for the kind of diversity we focus on in this chapter and related policy diversity works (Cully & Demiris 2018, Flageat et al. 2023, Zahavy et al. 2023). There are other aspects applications for which this definition might not be general enough, e.g., diverse environment design (Wang et al. 2019).

8.2.1 Context-conditioned Diversity Learning

Akin to Zahavy et al. (2023) we want to maximise diversity within our set of policies and express distances between policies in terms of expected features ψ (Borsa et al. 2019, Carvalho et al. 2023) – which are a function of state-occupancy. Hence, we measure ψ distances in accordance with Def. 8.1, and we use a context-based Hausdorff distance (Rockafellar 1970) to force context-dependant diversity on our set of policies Π_c . However, we do not wish to compromise too much optimality to get diversity. Specifically, we aim at training an RL agent that, given a context c , discovers a set of n policies $\Pi_c = \{\pi_c^i\}_{i=1}^n$ by successfully optimising:

$$\max_{\Pi_c} \text{Diversity}(\Pi_c) \text{ s.t. } d_{\pi_c} \cdot r_e \geq \rho \hat{v}_e, \quad \forall \pi_c \in \Pi_c \quad (8.1)$$

where d_{π_c} is our occupancy metric, r_e is the environment reward, that we refer to as *extrinsic* reward, ρ is a hyperparameter, and $\hat{v}_e$ is the value of a target policy. In practice, the target policy refers to a policy in our set Π^n , that ignores the diversity objective and is trained exclusively to maximise the extrinsic rewards. As in Zahavy et al. (2023), the policy value v expresses the expected reward accumulated by

the policy. To support the formalisation of our diversity objective, we define diversity as a distance over a set of occupancy metrics $\text{Diversity} : \left\{ \mathbb{R}^{|S||A||C|} \right\}^n \rightarrow \mathbb{R}$. Intuitively, such an objective is defined to both maximise the occupancy metrics and to preserve near-optimality of the set of policies. Policies are considered near-optimal if they achieve a value at least $\rho \hat{v}_e$, where $\rho \in [0, 1]$ is a hyperparameter of DUPLEX that we refer to as optimality ratio.

In order to promote diversity between the different policies we employ the repulsive rewards from Zahavy et al. (2023) and extend them to account for the context of the environment. Thus, at timestep t given the tuple (s_t, a_t, π^i, c) and cumulant ϕ , the diversity reward is:

$$r_d^i(s_t, a_t, c) = \phi(s_t, a_t, c) \cdot (\Psi_{\pi_c^i} - \Psi_{\bar{\pi}_c^i}) \quad (8.2)$$

where $\Psi_{\pi_c^i}$ are the expected features from policy π^i at t and $\bar{\pi}_c^i$ refers to the policy with the closest expected features to π^i in c sampled from our distribution μ_C . Intuitively, for each $c \sim \mu_C$, this reward encourages the algorithm to sample different state-action pairs than those occupied by the closest policy in c .

Stabilising diversity across different domains. Eq. 8.2 expands DOMiNO’s repulsive force to operate across different contexts. However, this reward alone is not enough to guarantee diversity since the scale of the intrinsic rewards will vary greatly through different environments, i.e., some domains will naturally yield more different successor features than others. Moreover, extrinsic rewards may also differ significantly through different settings and this would likely require noticeable reward engineering. Thus, aiming to make DUPLEX as domain-agnostic as possible, we introduce two novel components that modulate and constrain the intrinsic reward r_I . Namely, we introduce (1) a *dynamic intrinsic reward factor* χ that scales r_d to target a factor of the moving average of the general extrinsic value, and (2) a *soft-lower bound* λ that limits the search of diverse policies to a near-optimal subspace. Accordingly, we then define the DUPLEX intrinsic reward as:

$$r_I = \lambda \cdot \chi \cdot r_d \quad (8.3)$$

where χ scales r_d proportionally to $v_{e_{\text{avg}}} = \frac{1}{n} \sum_{i=1}^n v_{e_{\text{avg}}}^i$ with $v_{e_{\text{avg}}}^i = \alpha_{v_{\text{avg}}} v_{e_{\text{avg}}}^i + (1 - \alpha_{v_{\text{avg}}}) r_{e,t}^i$. Here,

$\alpha_{v_{\text{avg}}} \in [0, 1]$, and $r_{e,t}^i$ is the extrinsic reward at t when the agent acts under policy i . Hence, for every algorithm update, χ is updated as follows:

$$\chi_t = \alpha_\chi \chi'_t + (1 - \alpha_\chi) \chi_{(t-1)} \quad (8.4)$$

where α_χ is the update rate of χ , and $\chi' = |v_{e_{\text{avg}}}/v_{d_{\text{avg}}}|(1 - \rho)$ is the target value replacing r_d with r_e and making $v_{d_{\text{avg}}}$ equivalent to $v_{e_{\text{avg}}}$. χ minimises the domain dependency by having a user-defined optimality ratio ρ , which is a weighted distribution between the objectives of maximising extrinsic rewards and diversity, e.g., if $\rho = 0.8$, χ dynamically updates towards $v_{d_{\text{avg}}} = 0.2v_e^{\text{avg}}$. Moreover, in the experiment section, we demonstrate that Equation (8.4) provides an update rule that can be applied across different domains. However, we observed that a few policies would learn degenerated behaviors in order to maximise diversity. For instance, (in MuJoCo) some policies will learn different ways to stand still. To remove this possibility, we explored constrained optimisation but found it limiting in complex domains like GT. Thus, we include λ to bound the near-optimal subspace for each policy as:

$$\lambda = \left\{ \sigma_k \left(\frac{v_{e_{\text{avg}}}^i - \beta \hat{v}_{e_{\text{avg}}}}{|\hat{v}_{e_{\text{avg}}} + l|} \right) \right\}_{i=1}^n \quad (8.5)$$

where $\sigma_k(x) = 1/(1 + e^{-kx})$ and $\beta \in [0, 1]$ is a hyperparameter indicating the reward region we are interested in exploring, k regulates how “soft” is the bound, n is the number of policies, l is a small constant to prevent division by zero, and $v_{e_{\text{avg}}}$ is the average extrinsic value. Intuitively, λ limits the exploration of diverse behaviors to a near-optimality area defined by the threshold β . For k and α_χ , we found empirically a set of values that proved to be stable and we kept them unchanged through all domains and tasks.

Remark 5. Given k and α_χ are always fixed and domain agnostic. *DUPLEX* only requires the user to set two parameters: 1) what relationship they want between extrinsic and intrinsic rewards through the hyper-parameter ρ , e.g., with $\rho = 0.5$ extrinsic rewards should be around two times larger than the intrinsic rewards 2) at what level of suboptimality they are not interested on the policies to continue exploring, which is set through β , e.g., with $\beta = 0.5$ policies that achieve less than half the extrinsic reward that our target policy will discard any diversity reward.

Constrained optimisation. Within the algorithm framework, we allow Lagrangian constraint optimisation Borkar (2005) to be contingent to specific domains. In fact, when constrained optimisation is configured, we establish a multi-goal objective Altman (1999), Szepesvári (2020) using Lagrange multipliers κ , whose objective is to maximise $\kappa(\rho \hat{v}_e - d_\pi \cdot r_e)$. Intuitively, this objective yields a multiplier κ that decreases in value when the policy is satisfying the optimality constraint, and increases otherwise. However, we empirically find that while Lagrangian constrained optimisation might be beneficial in canonical domains, it is not effective in complex environments such as GT and fails to find competitive diverse behaviors. Nevertheless, Equation (8.4) and Equation (8.5) make our r_i less reliant on constraint optimisation than previous approaches. For these reasons, we implement constrained optimisation in DUPLEX by keeping κ fixed by default and make it learnable as an optional feature.

8.2.2 Estimating Successor Features

We rely on expected features ψ to determine policy diversity (Def. 8.1). Hence, it is fundamental to estimate ψ reliably. As we anticipated in Section 8.1, it is not always best to keep a ψ^{avg} for each context and task since they may be infinite. Thus, we build on top of USFA and focus on discounted expected features (SFs) $\psi^\gamma(s, a, z, c)$ and (similar to the version of DOMiNO with estimated features) incorporate an extra head in the critic to estimate SFs $\tilde{\psi}^\gamma(s, a, z, c)$. However, aligning with previous literature (Carvalho et al. 2023), we found that the critic struggles to correctly estimate ψ^γ in complex settings. Specifically, since intrinsic rewards are bootstrapped over the expected difference of successor features, stability in the estimation plays a major role. Thus, taking inspiration from Haarnoja et al. (2018), we incorporate an entropy term in the expected features objective to improve accuracy in the estimation of SFs. That is, for policy i in context c our SFs at s_t are:

$$\psi^{\gamma,i}(s_t, a_t, c) = \phi_t + \mathbb{E}_{\pi_c} \sum_{k=t+1}^{\infty} \gamma^{k-t} [\phi_k + \alpha_H H(\pi_c^i(\cdot | s, c))] \quad (8.6)$$

Where α_H is the entropy weight. Now we can formulate a temporal difference loss for the critic output $\tilde{\psi}^\gamma$ that accounts for the entropy in the policy:

$$\text{TD}_H(\tilde{\Psi}_{\theta_j}^\gamma(\pi_c^z)) = \mathbb{E}_{(s,a) \sim \pi_c^z} \left[\frac{1}{2} \left(\tilde{\Psi}_{\theta_j}(t) - y(\phi, s', z) \right)^2 \right] \quad (8.7)$$

with $t = s, a, c, z$. Note that, in a similar fashion to SAC, we have two critic estimates $j = \{1, 2\}$, but different from taking the min of the two, our target is obtained from the average of the estimates:

$$y(\phi, s', z) = \phi(t) + \gamma \left(\text{avg}_{j=1,2} \tilde{\Psi}_{\theta_{\text{arg},j}}^z(s', \tilde{a}'_z, c) - p_\omega^z(\tilde{a}'_z | s', c) \right) \quad (8.8)$$

with $p_\omega^z \equiv \alpha \log \pi_\omega^z$ and $\tilde{a}'_z \sim \pi_\omega^z(\cdot | s', c)$. The motivation behind averaging instead of taking the min lies in the different impact of overestimation on value and successor feature estimates. When predicting values, overestimation directly influences the policy since it is trained to maximise advantage. However, in DUPLEX, successor features assume a distinct role: the policy is not incentivised to select actions with the highest successor features but rather those that differ the most from the successor features of the remaining policies (subject to not sacrificing too much performance). Consequently, precision becomes of greater significance for Ψ^γ and taking the average emerges as a better solution.

The introduction of entropy improves stability in the estimation of SFs through most of the training process and, as we see below in empirical results, it supports OOD generalisation. However, as learning progresses, we typically encounter moments where the estimated difference between policies increases its order of magnitude rapidly. Such a change in the magnitude of expected distances leads to an undesired effect, where intrinsic rewards also scale rapidly, resulting in increasingly larger value estimate errors and divergence of the critics. To address such events, we modify the update rate of χ :

$$\alpha_\chi = \begin{cases} 1 & \text{if } |v_{d_{\text{avg}}}| > \epsilon_\chi |v_{e_{\text{avg}}}| \\ \alpha_{\text{default}} & \text{otherwise} \end{cases} \quad (8.9)$$

Intuitively, we want r_I to motivate the agent to explore increasingly diverse behaviors without them stagnating the extrinsic rewards. For this reason we need α_χ to be a very low value – we use $1e-5$ across all our experiments. An exception is made when a sudden drastic change in $\tilde{\Psi}$ could cause the critic to diverge. We automatically detect those changes through a threshold $\epsilon_\chi |r_e^{\text{avg}}|$, where ϵ_χ is a scalar, and perform a one-step update to prevent r_I from destabilising the learning process. Similarly

Algorithm 6 DUPLEX

-
- 1: Initialize dataset with random exploration, set up critics θ and policy parameters ω . and initialize moving averages $\{v_{\text{avg}}^{\pi^i}\}_{i=1}^n = 0$ reward weight $\kappa = 0$, $\chi = 1$ and step = 0
 - 2: **while** not converged **do**
 - 3: Generate more data samples
 - 4: Sample batch of m transitions $\{t_j\}_{j=1}^m$, where a transition is $t_j = (s^j, a^j, c^j, \phi(s^j, a^j, c^j), z^j, r_e^j, s'^j)$
 - 5: Use critic to get $\tilde{v}_{e_j}^{\pi^i}$, $\tilde{v}_{i_j}^{\pi^i}$ and $\tilde{\Psi}^{\gamma}(s, a, c, z)$ for every $\pi^z \in \Pi$. Where $\tilde{\Psi} = \text{avg}_{h=1,2} \tilde{\Psi}_{\theta_h}$
 - 6: Compute $r_I(s)$ from $\tilde{\Psi}, z, \phi(s, a, c)$ with Equation 8.3
 - 7: Compute extrinsic temporal difference (TD) errors and q-values: TD_e, Q_e
 - 8: For sampled z get TD error with entropy on successor features with Equation 8.7: TD_H
 - 9: Compute intrinsic TD errors and q-values: TD_I, Q_I
 - 10: Combine q-values $Q = \sigma(\kappa^i)Q_e + (1 - \sigma(\kappa^i))Q_I$
 - 11: **if** step > critic warming-up period **then**
 $\triangleright$ Policy loss:
 - 12:
$$\nabla_{\omega} \sum_j \left(\min_{h=1,2} Q_{\theta_h}(s, \pi_{\omega}(\cdot | s^j, c^j, z^j)) - \alpha \log \pi_{\omega}(\pi_{\omega}(\cdot | s^j, c^j, z^j) | s^j, c^j, z^j) \right)$$
 - 13: Calculate weighted critic loss: $b_v(\text{TD}_e + \text{TD}_i) + b_{\Psi}(\text{TD}_H)$
 - 14: **if** Constrained optimization **then**
 $\triangleright$ Lagrange loss
 - 15:
$$\sum_{i=1}^n \sigma(\kappa^i) (v_{\text{avg}}^i - \rho \hat{v}_{\text{avg}})$$
 - 16: Update κ
 - 17: Update θ and ω
 - 18: Update $v_{e_{\text{avg}}}^i = \alpha_{v_{\text{avg}}} v_{e_{\text{avg}}}^i + (1 - \alpha_{v_{\text{avg}}}) r_{e,t}^i$, $v_{d_{\text{avg}}}^i = \alpha_{v_{\text{avg}}} v_{d_{\text{avg}}}^i + (1 - \alpha_{v_{\text{avg}}}) r_{d,t}^i$, χ with Eq. 8.4 and step + = 1
-

to α_{χ} once we found a stable value for ϵ_{χ} we left this parameter fixed across all experiments and environments.

8.2.3 Training DUPLEX

DUPLEX follows an actor-critic approach, where a critic network estimates values and, in the case of multiple contexts, successor features, while the policy network decides which action to take. Figure 8.1 depicts the workflow of the algorithm. The networks have three independent input encoders for the state, policy and context respectively. Since we use SAC (Haarnoja et al. 2018) as our base RL algorithm, we employ two critic networks and take the min of the two for value estimates $\tilde{v}^{\gamma}$ and the average for $\tilde{\Psi}^{\gamma}$ (Eq. 8.8). The training procedure of DUPLEX is described in Line 6. After warming up to collect data, we start sampling transitions, t , to learn (Line 4). Since we need to estimate the distance between all policies to infer the intrinsic rewards, we broadcast the input of the critic

to have the SF estimates for all policies in every transition and do a forward pass through the critic (Line 5). Then, we obtain r_I with Eq. 8.3 and compute the various temporal-difference errors (Line 7 – Line 9). Similarly, after a short warm-up period, we also compute the policy losses (Line 12). Specifically, we compute losses by calculating the advantage from the combination of extrinsic and intrinsic rewards. Then, depending on if we exploit constrained optimisation, we update the Lagrange multipliers (Line 15), and finally, we update the the network weights and the average policy values in order to adjust the dynamic factor χ (Line 17 and Line 18).

8.3 Experiments

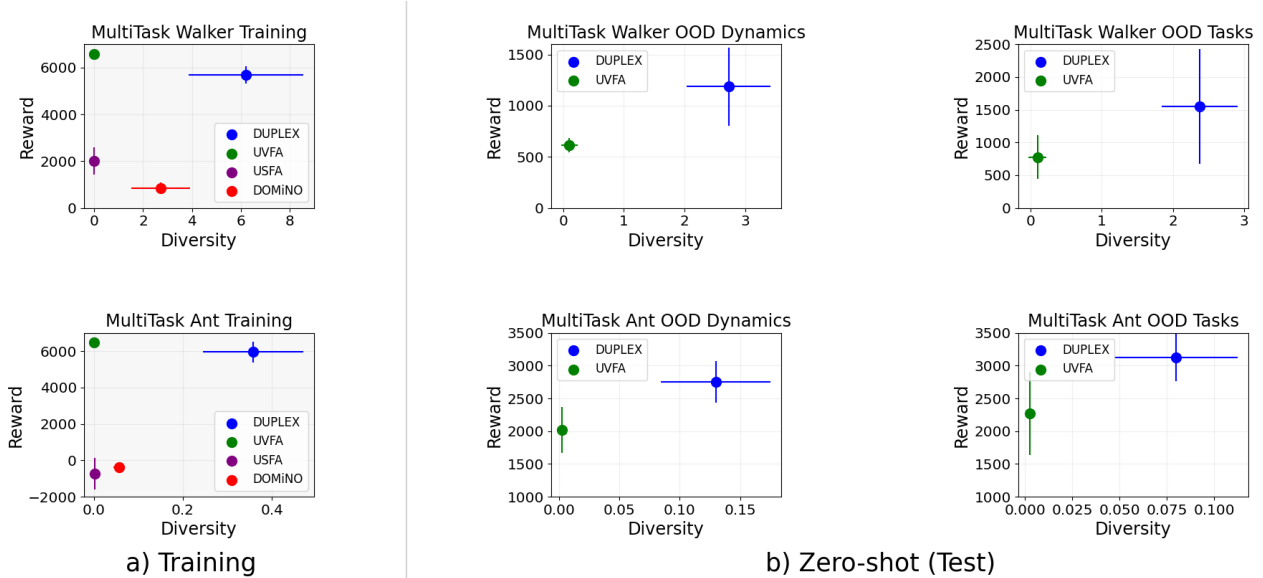


Figure 8.2: Results in Mujoco Walker (Top) and Ant (bottom) multitask environments. On the y-axis we report the reward accumulated by the agent while on the x-axis diversity according to Def. 8.1. Ideally, we want to be as close as possible to the top right region. OOD dynamics refers to gravities that are at least 40% stronger or weaker than the strongest and weakest gravities seen in training, respectively. OOD tasks refers to tasks where the agent needs to walk forward below a height 20% lower than the lowest height seen in training tasks. Dots and lines represent mean and standard deviation, respectively. Note that only DUPLEX and UVFA are evaluated in Test since the other algorithms already fail at training.

Our experiments are designed to evaluate whether DUPLEX (1) improves on previous baselines in learning diverse near-optimal policies (Figure 8.4); (2) is effective in different contexts while generalising to OOD dynamics and tasks (Figure 8.2); (3) improves on previous baselines in estimating SFs (Figure 8.5 and C.1); and (4) is successful in learning diverse driving styles in a realistic physics

simulator such as GT (Figure 8.3). All results in Mujoco are obtained from five independent runs. GT experiments are more computationally demanding and the results are obtained from three different runs. Videos are included at <https://sites.google.com/view/bgl-phd-duplex/home>.

Environments. Our core experiments are performed in multitask versions of Mujoco’s Walker and Ant environments. We extend the default versions of these scenarios to include a three-dimensional context c where the inputs represent: gravity, the upper and lower height thresholds we would like the agent to respect. We also perform ablations such environments with a fixed context, standard gravity and only requiring the agent to move forward. This allows us compare DUPLEX with DOMiNO employing ψ^{avg} as well. Last, we evaluate DUPLEX in a GranTurismo track, where the goal is to have different driving styles while completing laps fast. Achieving diverse and competitive driving styles in such realistic and complex simulator Wurman et al. (2022) can unlock multiple applications both in gaming and self-driving. In GT and default Mujoco environments, where there is a single context and task, we use ψ^{avg} , while in the multitask scenarios we employ ψ^{γ} . Additional details about the metrics and the environments are reported in Appendix C.1.

Results in Multitask Environments. Figure 8.2 (left column) illustrates the results in multitask scenarios of the different approaches when evaluated in a within distribution setting. Here only DUPLEX provides both good performance and diversity. UVFA struggles to provide diversity while achieving good task performance and, both USFA and DOMiNO fail at learning any competitive policy and at providing diversity. DUPLEX, in fact, succeeds at learning satisfactory behaviors since it stabilises estimation of SFs and more robustly modulates the intrinsic reward (Eq. 8.3). Additionally, note that USFA and UVFA do not optimise for diversity and their diversity score is expected to be close to zero.

Similar analysis can be conducted in the middle and right column in Figure 8.2. There we evaluate the effectiveness of our approach, and UVFA, when operating out-of training distribution. Note that the USFA and DOMiNO baselines do not manage to qualify in this setting as their performance in OOD contexts proved to be highly unsatisfactory – and we could not make them succeed in completing the task. That said, we observe that in all the OOD scenarios DUPLEX outperforms UVFA and, most

importantly, it covers a bigger range of diversity values while completing the tasks.

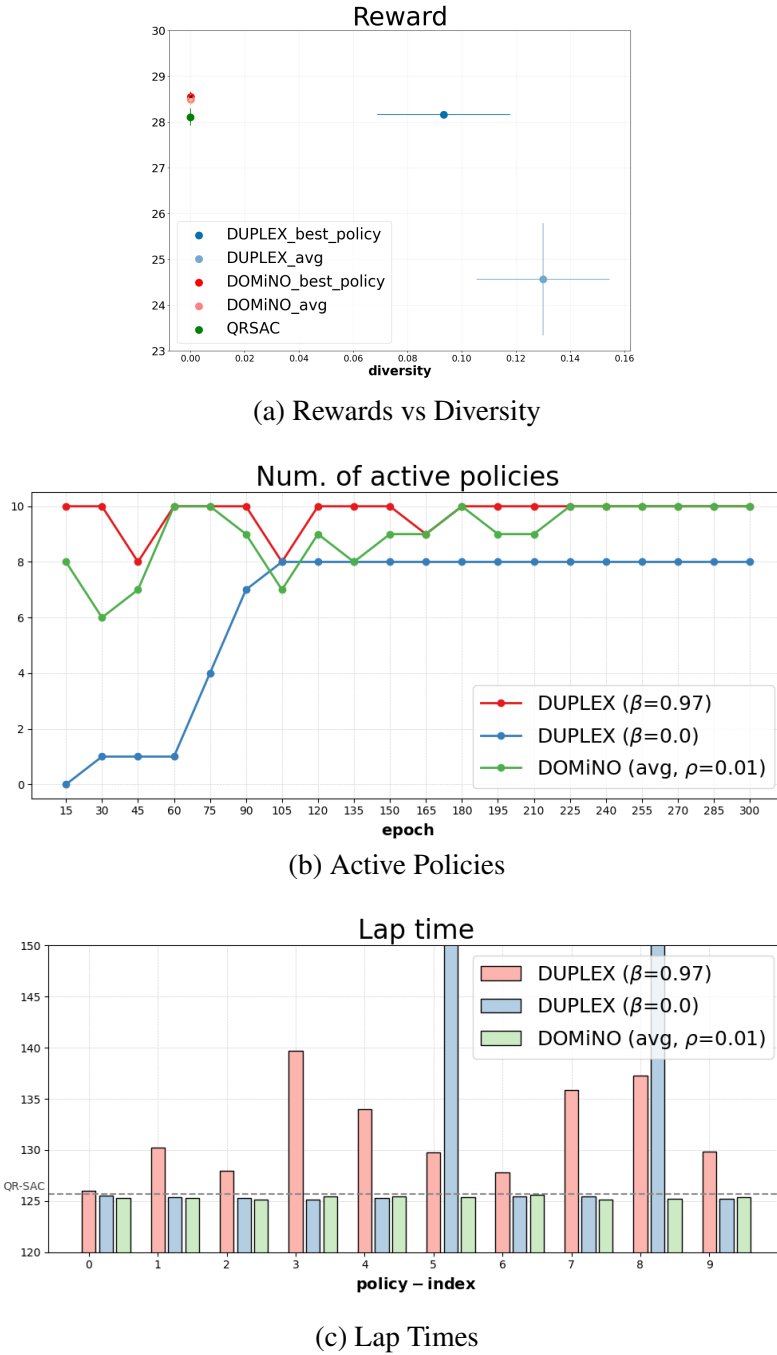


Figure 8.3: Results in GT. Top is the reward-diversity trade off, DUPLEX/DOMiNO_best_policy $\max_{\pi \in \Pi}$ of every run, while DUPLEX/DOMiNO_avg refers to $\text{avg}_{\pi \in \Pi}$ of that run. We see that only DUPLEX is able to generate *both* competitive and diverse behaviors. The remaining figures illustrate the number of active policies (middle), i.e., number of policies that are finishing laps, and min lap times of the diverse policies (bottom). Note that the number of active policy is important to showcase that all policies are actively searching for diverse behaviors through different epochs. In fact, our target is having all policies active with diverse lap times (diversity) below the 150 seconds mark (near-optimality). We find that applying our soft lower bound ($\beta \neq 0$) is key to focus diversity in the region of interest.

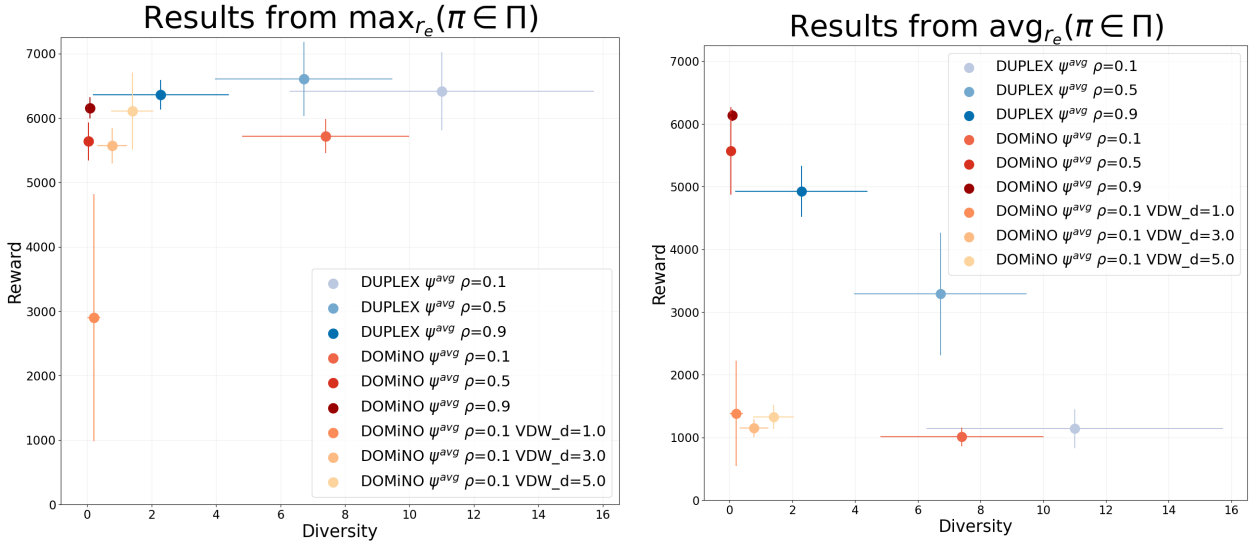


Figure 8.4: In a Walker single-task settings, both DUPLEX and DOMiNO do not estimate SFs and use $d_{\pi_c}^{\text{avg}}$ instead. On the left, we show results from the best policy in each variant while on the right, the average reward across all policies.

Results in GranTurismo We test the ability of DUPLEX to provide competitive diverse behaviors in the high-end driving simulator of GT (e.g., different lap-time, driving lines and/or braking points). As in Wurman et al. (2022) we use QRSAC as the base RL algorithm for both DOMiNO and DUPLEX. Figure 8.3 (left) illustrates that DUPLEX is the only framework able to provide highly diverse rewards in the near-optimal region. Moreover, Figure 8.3 illustrates how this diversity translates into varied lap times. We also include an ablation to provide evidence that, in such complex environments, our lower bound β formulation is key to support diversity within the near-optimality. We see that only DUPLEX with $\beta > 0$ trains policies that complete laps within 150 seconds and that also guarantee diversity – opposing to DOMiNO that collapses all policies to a similar behavior even when using an extremely flexible constraint objective. That is because, when using $\beta = 0$, the most diverse policies are “busy” finding different ways to *not* finish the track. In the case of DOMiNO, we observe that it fails to provide diversity even if it implements a constraint optimisation objective that should support more adaptability. This is explained by the fact that, even if DOMiNO with $\rho = 0.01$ is configuring the learning algorithm to consider as “acceptable” any diverse policy achieving 1% of $\hat{v}_{e_{\text{avg}}}$ (i.e., out of the optimality region), the algorithm still struggles to find diverse policies (as showed in the figure). Such a result exposes the limitations of using Lagrangian multipliers in this complex domain.

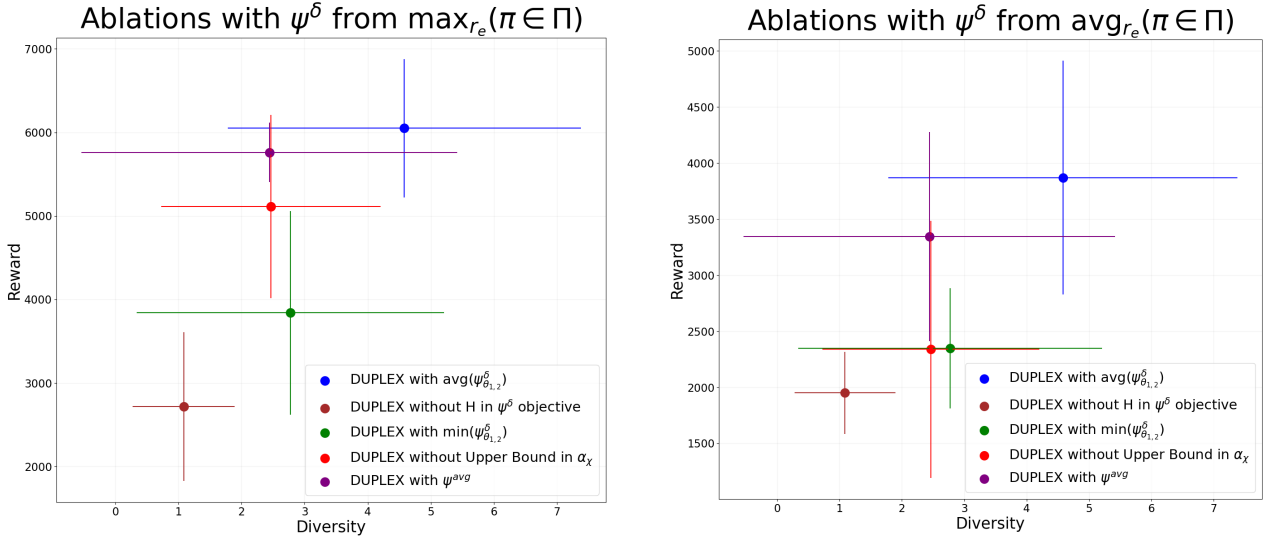


Figure 8.5: Ablation of DUPLEX in Multitask Walker when 1) not using entropy (H) in the ψ^δ objective from Eq. 8.7; 2) using $\min(\psi_{\theta_{1,2}}^\delta)$ - as in SAC - instead of $\text{avg}(\psi_{\theta_{1,2}}^\delta)$; 3) eliminating the upper bound trigger in α_χ ; 4) using ψ^{avg} instead of ψ^δ .

Ablation of diversity. To further demonstrate the benefits in diversity of DUPLEX, we include a comparison of DOMiNO and our framework with ψ^{avg} in the default Walker2D. DOMiNO was specially designed to work with ψ^{avg} , and in this ablation we can see the benefits of our r_I (Eq. 8.3) isolated from the impact of estimating successor features. Figure 8.4 includes the results from both frameworks with multiple optimality ratios ρ . We configure DOMiNO to also explore the Van Der Wall (VDW) distance hyperparameter, which includes an attraction force used to define a desired distance between all the policies in the set (see Zahavy et al. (2023) for details). We find that DUPLEX is able to find a better Pareto-frontier for the quality-diversity objective (Figure 8.4 right), while achieving better optimal policies than DOMiNO (Figure 8.4 left).

Ablation of successor-feature estimation. Figure 8.5 illustrates the impact of employing ψ^δ and the different features incorporated by DUPLEX. We see that all are enabling components and that accounting for the entropy of the policies in Eq. 8.7 when estimating SFs is key. Notably, we find that this feature is also greatly beneficial to existing frameworks such as USFA (see Appendix C.2).

8.4 Discussion

DUPLEX provides a powerful technique that enables learning of diverse near-optimal behaviors in realistic complex domains (such as GranTurismo), and in challenging multi-task Mujoco environments. Results in the multi-task settings show that DUPLEX is the first RL framework that can (i) learn effective diverse behaviors in complex environments and (ii) generalise to OOD environment dynamics and unseen task variations.

DUPLEX opens up opportunities to explore various research directions focused on the limitations of the current framework. For example, how can we scale our framework to learn hundreds of policies? How can we improve on sample efficiency and keep a low computational footprint while learning diversity? Can we control in what measure a particular policy is going to be different? Can we combine diverse policies to enhance zero-shot transfer to different tasks and contexts? This latter question is very interesting to us. In fact, we suppose that once diverse policies have been learned, each of them retains different useful skills that can be combined dynamically to reconstruct a single agent capable of acting optimally and tackling unseen portions of the state-space. In earlier version of DUPLEX we explored the generalised policy improvement (GPI) algorithm Barreto et al. (2017), Borsa et al. (2019) to merge the strengths of the different policies. However, in our preliminary experiments we found that the final GPI policy is brittle when compared to the ones generated directly by the policy network. We remark that so far, GPI has only proved successful in toy and discrete action environments. Thus, we hypothesised that GPI fails to accurately estimate the critic value in complex, continuous-action scenarios like the ones we used in this chapter. Finally, the presented approach holds significant promise for advancing research in diverse policy learning, scalability, and efficient transfer across varied tasks and contexts, which can yield autonomous agents that effectively operate in safety-aware settings.

Chapter 9

Conclusions and Future Work

This thesis addresses a number of fundamental limitations in state-of-the-art reinforcement learning, with a particular emphasis on shortcomings in safety-aware applications. The work contributes in three distinct areas: 1) the application of formal logic to DRL (Chapters 4 and 5), 2) systematic and out-of-distribution generalisation in reinforcement learning (Chapters 6 and 7), and 3) Robustness and diversity of autonomous agents in racing and robotics simulators (Chapter 8).

Specifically, Chapters 4 and 5 alleviated the need of single-agent and fully-observable formalism, respectively, that was previously existing in the literature combining formal methods and reinforcement learning. Specifically, Chapter 4 introduced the extended Markov games (EMGs) a new mathematical model to work with RL algorithms in non-Markovian multi-agent settings and demonstrated how they work empirically in Section 4.4. Then, Chapter 4 expanded this research further by presenting extended partially-observable markov decision processes (EPOMDP) a new model to handle non-markovian specifications in partially observable settings through RL. Future work combining formal methods and RL can rely on these formulations to naturally describe complex formal specifications that are suited for RL algorithms with more realistic assumptions.

Chapters 6 and 7 presented novel inductive biases and a deep learning architecture to improve systematic and out-of-distribution (OOD) generalisation in DRL. In Chapter 6, we showed that, contrary to previous state-of-the-art claims that any DRL agent trained from scratch requires 100 different

instructions to generalise systematically to logical operators such as negation (Hill et al. 2020), it can be done with as few as six instructions. We provided empirical evidence in diverse, pixel-based grid worlds that the combination of 1) a convolutional encoder with the right architecture and 2) the correct evaluation setup, could overcome difficulties that hindered earlier studies. Specifically, we discovered that having independent object-oriented representations in the visual encoder facilitates systematic learning. Moreover, we found that DRL agents are prone to overfit negated training instructions by applying the negation only to the training objects and ignoring the negation operator on test instructions. We also observed that, prior to learning with confidence to choose the right object, agents tend to not select any object at all, a form of overfitting that misled previous research (Hill et al. 2020). In our case, through a reliable-deceptive evaluation setup and by incorporating a train-validation-test data split, we detected this phenomenon and found evidence of emergent generalisation. Hence, our work highlights the need in RL to standardise the use of robust evaluation systems as in other research fields, e.g., supervised learning, to minimise incorrect hypotheses in future works. Lastly, Chapter 6 also presents a novel neuro-symbolic framework that leverages the strengths of symbolic AI to generalise to compound instructions while reinforcement learning masters the atomic goals. To our knowledge, this is the first DRL framework that is able to execute any formula on a given formal language while relying on a single neural network. We remark that until our work in Chapter 6, all neuro-symbolic frameworks had either focused on solving training tasks (Icarte et al. 2018, Hasanbeig et al. 2021, Hammond et al. 2021) or relied on training from scratch new policy networks when tackling novel instructions (Kuo et al. 2020, Icarte et al. 2019), with its associated computational burden.

Chapter 7 further advances the state-of-the-art in DRL generalisation with a novel deep learning configuration: the latent-goal architecture. This architecture is designed with the aim of training agents that can successfully execute zero-shot, i.e., unseen, instructions in out-of-distribution (OOD) scenarios. In our experiments OOD scenarios included challenges such as navigating through maps significantly larger – ~ 4 times larger – than those encountered during training. The latent goal architecture operates on the premise that instructions are provided through a separate input channel and builds on two foundational ideas: creating a task-agnostic representation of the input and integrating this with a goal generator. This generator processes both instruction and observation, and must fun-

nel the information through an information bottleneck. The benefits of latent-goal architectures were demonstrated with various configurations, ranging from simple MLPs (Goodfellow et al. 2016) and relational networks (Santoro et al. 2017), to state-of-the-art attention-based architectures like MHA (Zambaldi et al. 2018) and PrediNet (Shanahan et al. 2020), as well as state-of-the-art hierarchical architectures such as BRIMS (Mittal et al. 2020). Remarkably, the latent goal configuration enhanced the performance of each network tested, without the need for any fine-tuning. Furthermore, Chapter 7 presents detailed empirical evidence showing that the integration of latent-goal architectures with layers that utilise both attention and relational reasoning, exemplified by PrediNet, resulted in a 33% and 22% improvement in performance over the best conventional architectures in the most challenging OOD scenarios of the Minecraft-inspired and Minigrid benchmarks, respectively (see Section 7.3).

Building on our advances in state-of-the-art DRL generalisation, future work could investigate how the advantages of object-oriented representations and latent-goal architectures scale up to larger datasets and deeper neural network configurations, such as transformers (Chen et al. 2021) or state-space models (Gu & Dao 2023). We anticipate that object-oriented representations will naturally emerge with sufficiently diverse training data (Hill et al. 2021). However, the specific benefits introduced by the biases present in latent-goal architectures may not occur naturally in other configurations and could offer an advantage to future large-scale models that utilise latent goals. Research carried out in parallel to the work of Chapter 6 and 7 showed that DRL agents exhibit strong generalisation capabilities when augmented with external memory (Hill et al. 2021, Lampinen et al. 2021). Yet, existing solutions are constrained by short context windows. Future research should consider merging the benefits of latent-goal architectures with external memory, alongside incorporating auxiliary forgetting mechanisms to facilitate continual learning scenarios (Bhattacharjee & Mahajan 2022). Additionally, our work in Part II utilised straightforward symbolic frameworks, which future studies could evolve towards more intricate language manipulation tasks, such as determining the optimal subtask sequence to fulfill overall objectives, as demonstrated in Vaezipoor et al. (2021).

Finally, Chapter 8 develops agents capable of exhibiting diverse and competitive behaviors in challenging safety-aware benchmarks, including GranTurismoTM 7 (GT) (Wurman et al. 2022), a hyper-realistic driving simulator. We tested previous state-of-the-art efforts Zahavy et al. (2023, 2024) but

they struggled to discover diverse policies without resulting in the vehicle crashing (Section 8.3). Consequently, Chapter 8 introduced DUPLEX, a groundbreaking framework for multitask policy diversity that utilises universal estimators (Schaul et al. 2015, Borsa et al. 2019) and innovative mechanisms to facilitate robust diversity learning in RL. Specifically, DUPLEX integrates methods to enhance stability and usability, enabling users to simply determine the balance between the diversity objective and the extrinsic goal as well as the region of interest for the agents to explore, e.g., what minimum average speed should the different driving styles have in GT. Furthermore, in Section 8.2.2, we presented a novel optimisation objective that enables DUPLEX to generalise to OOD tasks and environments while maintaining diverse behaviors under such conditions. Our empirical analysis, conducted in GT and Mujoco, demonstrates the robust performance of our framework. Additionally, we have included videos at <https://sites.google.com/view/duplex-submission148/home>, allowing readers to visualise the performance and diversity achieved by DUPLEX.

A limitation of DUPLEX is its inability to merge the strengths of various policies into a single, improved policy. While we investigated potential solutions such as generalised policy improvement (GPI) (Barreto et al. 2017), this method proved inapplicable within the context of function approximators (deep learning) and continuous action spaces. Therefore, future research could focus on discovering innovative approaches to enhance the robustness of DUPLEX in these settings. Another promising direction involves the use of a descriptive policy encoder. Specifically, equipping DUPLEX with information on what relevant features a particular policy should possess could significantly influence the policy discovery process, granting more control over it. This approach was examined in parallel research (Grillotti et al. 2023), indicating that successor features (refer to Section 8.1) hold promise as a tool for achieving controllable policy discovery.

In sum, this thesis identified key weaknesses in DRL that affected the deployment of this framework in safety-aware scenarios. These weaknesses included brittle generalisation, poor robustness and policy exploration, and unrealistic assumptions in formalism combining RL and temporal logic. We did key contributions to tackle those shortcomings through novel solutions that improved systematic and OOD generalisation, policy diversity, and proposed more general formalism. Our work accomplishes this by contributing with groundbreaking findings - the right biases for the emergence

of systematic generalisation with logic operators, new mechanisms for accomplishing stable diverse exploration and learning in challenging driving scenarios-, novel deep learning architectures -the latent-goal architecture-, frameworks -DUPLEX, and the neurosymbolic frameworks in Part II-, formal languages -TTL and SATTL-, and mathematical models - EMGs and EPOMDPs-.

9.1 Formal Languages and Systematic Generalisation with LLMs

Taking a broader perspective, the lines of work presented in this thesis can be highly relevant for future advancements in AI and in the emergent field of Large Language Models (LLMs) (Brown et al. 2020, OpenAI et al. 2024).

Formal methods can play a significant role in the integration of large language models (LLMs) into safety-aware applications. Similar to their use in reinforcement learning (RL) to ensure safety and correctness, formal methods can provide specifications against which LLMs must contrast to progress effectively. For example, in an LLM-powered non-player character (NPC) within a video game, the game directors may create a sequence of scenes requiring specific lines for the game to advance. By integrating formal methods, the LLM's generated content can be aligned with these predefined specifications, offering a mechanism for verification and controlled progression, much like the approaches discussed in Chapters 4 and 6.

Nevertheless, while Chapter 1 evidenced that LLMs still struggle with compositional reasoning in text, ongoing research in text-based reasoning may soon alleviate this challenge (Zelikman et al. 2024). As advancements are made in this area, the AI community will focus on the recurrent promise of generally intelligent autonomous systems (Wang et al. 2024), requiring to pivot back towards working with embodied settings. Thus, leading to a resurgence of interest in the recurrent topic of compositional and systematic generalization capabilities of embodied AI.

Last, despite the significant momentum of generative AI Touvron et al. (2023), Shanahan et al. (2023), RL algorithms are anticipated to remain useful in locally deployed simulators such as video games, low latency domains such as driving, or in reasoning and planning intensive tasks (?). Zooming in the

former, hardware limitations and computational costs make foreseeable that future games will feature LLM-powered conversational NPCs that employ RL for locomotion and plan execution. Moreover, diversity techniques are expected to become increasingly important, given their natural fit for providing adaptable solutions and generating rich, diverse data that can enhance robustness towards what humans might say or do (Samvelyan et al. 2024).

Combining these insights, it is possible to envisage systems where formal methods ensure alignment with specifications, LLMs And RL overcome challenges in compositional reasoning, and diversity techniques enforces robustness and adaptability in the conversations and behaviors of such autonomous systems. As computational resources continue to become more accessible, these systems have the potential to scale and evolve significantly. In theory, if data can be automatically collected from sufficiently rich sources, this process could be unbounded, with models co-evolving indefinitely by continually challenging (through diversity) and improving upon each other on skill (RL) and reliability (formal methods).

Bibliography

- Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N. & Riedmiller, M. A. (2018), Maximum a posteriori policy optimisation, in ‘6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings’, OpenReview.net.
URL: <https://openreview.net/forum?id=S1ANxQW0b>
- Achiam, J., Held, D., Tamar, A. & Abbeel, P. (2017), Constrained policy optimization, in ‘International conference on machine learning’, PMLR, pp. 22–31.
- Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S. & Topcu, U. (2018), Safe reinforcement learning via shielding, in ‘Proceedings of the AAAI Conference on Artificial Intelligence’, Vol. 32.
- Altman, E. (1993), ‘Asymptotic properties of constrained markov decision processes’, *Zeitschrift für Operations Research* **37**, 151–170.
- Altman, E. (1999), *Constrained Markov decision processes*, Vol. 7, CRC press.
- Anderson, G., Verma, A., Dillig, I. & Chaudhuri, S. (2020), Neurosymbolic reinforcement learning with formally verified exploration, in ‘Advances in Neural Information Processing Systems’, Vol. 33, pp. 6172–6183.
URL: <https://proceedings.neurips.cc/paper/2020/file/448d5eda79895153938a8431919f4c9f-Paper.pdf>
- Andreas, J., Klein, D. & Levine, S. (2017), Modular multitask reinforcement learning with policy

sketches, in ‘Proceedings of the 34th International Conference on Machine Learning-Volume 70’, JMLR. org, pp. 166–175.

Andreas, J., Klein, D. & Levine, S. (2018), Learning with latent language, in ‘Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)’, pp. 2166–2179.

Ardon, L., Furelos-Blanco, D. & Russo, A. (2023), Learning reward machines in cooperative multi-agent tasks, in ‘International Conference on Autonomous Agents and Multiagent Systems’, Springer, pp. 43–59.

Bacchus, F., Boutilier, C. & Grove, A. (1996), Rewarding behaviors, in ‘Proceedings of the National Conference on Artificial Intelligence’, pp. 1160–1167.

Bacchus, F. & Kabanza, F. (2000), ‘Using temporal logics to express search control knowledge for planning’, *Artificial intelligence* **116**(1-2), 123–191.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D. & Blundell, C. (2020), ‘Agent57: Outperforming the atari human benchmark’, **119**, 507–517.

Bahdanau, D., Cho, K. & Bengio, Y. (2015), Neural machine translation by jointly learning to align and translate, in Y. Bengio & Y. LeCun, eds, ‘3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings’.

Bahdanau, D., Murty, S., Noukhovitch, M., Nguyen, T. H., de Vries, H. & Courville, A. C. (2019), Systematic generalization: What is required and can it be learned?, in ‘7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019’, OpenReview.net.

Baier, C. & Katoen, J.-P. (2008), *Principles of model checking*, MIT press.

Baldassarre, F. & Azizpour, H. (2022), Towards self-supervised learning of global and object-centric representations, in ‘ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality’.

- Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P. & Silver, D. (2017), ‘Successor features for transfer in reinforcement learning’, *Advances in neural information processing systems* **30**.
- Barron, A. R. (1993), ‘Universal approximation bounds for superpositions of a sigmoidal function’, *IEEE Transactions on Information theory* **39**(3), 930–945.
- Barto, A. & Duff, M. (1993), ‘Monte carlo matrix inversion and reinforcement learning’, *Advances in Neural Information Processing Systems* **6**.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R. et al. (2018), ‘Relational inductive biases, deep learning, and graph networks’, *arXiv preprint arXiv:1806.01261* .
- Belardinelli, F., León, B. G. & Malvone, V. (2022), Enabling markovian representations under imperfect information, in ‘14th International Conference on Agents and Artificial Intelligence (ICAART 2022)’, Vol. 2, SCITEPRESS-Science and Technology Publications, pp. 450–457.
- Bengio, Y. (2017), ‘The consciousness prior’, *CoRR* **abs/1709.08568**.
- Bhattacharjee, R. & Mahajan, G. (2022), Learning what to remember, in ‘International Conference on Algorithmic Learning Theory’, PMLR, pp. 70–89.
- Bishop, C. M. (2006), *Pattern recognition and machine learning*, springer.
- Borkar, V. S. (2005), ‘An actor-critic algorithm for constrained markov decision processes’, *Systems & control letters* **54**(3), 207–213.
- Borsa, D., Barreto, A., Quan, J., Mankowitz, D. J., van Hasselt, H., Munos, R., Silver, D. & Schaul, T. (2019), Universal successor features approximators, in ‘7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019’, OpenReview.net.
URL: <https://openreview.net/forum?id=SlVWjiRcKX>
- Brafman, R. I., De Giacomo, G. & Patrizi, F. (2018), Ltlf/ldlf non-markovian rewards, in ‘Thirty-Second AAAI Conference on Artificial Intelligence’.

- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. et al. (2020), ‘Language models are few-shot learners’, *Advances in neural information processing systems* **33**, 1877–1901.
- Camacho, A., Icarte, R. T., Klassen, T. Q., Valenzano, R. & McIlraith, S. A. (2019), Ltl and beyond: Formal languages for reward function specification in reinforcement learning, in ‘Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)’, pp. 6065–6073.
- Canese, L., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M. & Spanò, S. (2021), ‘Multi-agent reinforcement learning: A review of challenges and applications’, *Applied Sciences* **11**(11), 4948.
- Cao, Y., Yu, W., Ren, W. & Chen, G. (2012), ‘An overview of recent progress in the study of distributed multi-agent coordination’, *IEEE Transactions on Industrial informatics* **9**(1), 427–438.
- Carr, S., Jansen, N., Junges, S. & Topcu, U. (2023), Safe reinforcement learning via shielding under partial observability, in ‘Proceedings of the AAAI Conference on Artificial Intelligence’, Vol. 37, pp. 14748–14756.
- Carvalho, W., Filos, A., Lewis, R. L., Lee, H. & Singh, S. (2023), Composing task knowledge with modular successor feature approximators, in ‘The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023’, OpenReview.net.
URL: <https://openreview.net/pdf?id=DrtSx1z40Ib>
- Carvalho, W., Lampinen, A. K., Nikiforou, K., Hill, F. & Shanahan, M. (2021), ‘Feature-attending recurrent modules for generalization in reinforcement learning’, *CoRR* **abs/2112.08369**.
URL: <https://arxiv.org/abs/2112.08369>
- Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A. & Mor-datch, I. (2021), ‘Decision transformer: Reinforcement learning via sequence modeling’, *Advances in neural information processing systems* **34**, 15084–15097.
- Chen, Z. & Brown, E. N. (2013), ‘State space model’, *Scholarpedia* **8**(3), 30868.

- Chevalier-Boisvert, M., Willems, L. & Pal, S. (2018), ‘Minimalistic gridworld environment for openai gym’, <https://github.com/maximecb/gym-minigrid>.
- Cho, K., van Merriënboer, B., Bahdanau, D. & Bengio, Y. (2014), On the properties of neural machine translation: Encoder-decoder approaches, in D. Wu, M. Carpuat, X. Carreras & E. M. Vecchi, eds, ‘Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014’, Association for Computational Linguistics, pp. 103–111.
URL: <https://aclanthology.org/W14-4012/>
- Chomsky, N. & Lightfoot, D. W. (2002), *Syntactic structures*, Walter de Gruyter.
- Christopher, J. (1992), ‘Watkins and peter dayan’, *Q-Learning. Machine Learning* **8**(3), 279–292.
- Chua, K., Calandra, R., McAllister, R. & Levine, S. (2018), ‘Deep reinforcement learning in a handful of trials using probabilistic dynamics models’, *Advances in neural information processing systems* **31**.
- Cobbe, K., Klimov, O., Hesse, C., Kim, T. & Schulman, J. (2019), Quantifying generalization in reinforcement learning, in K. Chaudhuri & R. Salakhutdinov, eds, ‘Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA’, Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 1282–1289.
- Cohen, M. H., Serlin, Z., Leahy, K. & Belta, C. (2023), ‘Temporal logic guided safe model-based reinforcement learning: A hybrid systems approach’, *Nonlinear Analysis: Hybrid Systems* **47**, 101295.
- Corsi, D., Marzari, L., Pore, A., Farinelli, A., Casals, A., Fiorini, P. & Dall’Alba, D. (2023), ‘Constrained reinforcement learning and formal verification for safe colonoscopy navigation’, *arXiv preprint arXiv:2303.03207*.
- Cully, A., Clune, J., Tarapore, D. & Mouret, J.-B. (2015), ‘Robots that can adapt like animals’, *Nature* **521**(7553), 503–507. arXiv:1407.3501 [cs, q-bio].
URL: <http://arxiv.org/abs/1407.3501>

- Cully, A. & Demiris, Y. (2018), ‘Quality and diversity optimization: A unifying modular framework’, *IEEE Transactions on Evolutionary Computation* **22**(2), 245–259.
- Cybenko, G. (1989), ‘Approximation by superpositions of a sigmoidal function’, *Mathematics of control, signals and systems* **2**(4), 303–314.
- Czarnecki, W. M., Gidel, G., Tracey, B., Tuyls, K., Omidshafiei, S., Balduzzi, D. & Jaderberg, M. (2020), ‘Real world games look like spinning tops’, *Advances in Neural Information Processing Systems* **33**, 17443–17454.
- De Giacomo, G., Iocchi, L., Favorito, M. & Patrizi, F. (2018), ‘Reinforcement learning for ltlf/ldlf goals’, *arXiv preprint arXiv:1807.06333* .
- De Giacomo, G., Iocchi, L., Favorito, M. & Patrizi, F. (2019), Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications, in ‘Proceedings of the International Conference on Automated Planning and Scheduling’, Vol. 29, pp. 128–136.
- De Giacomo, G. & Vardi, M. Y. (2013), Linear temporal logic and linear dynamic logic on finite traces, in ‘Twenty-Third International Joint Conference on Artificial Intelligence’.
- Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D. et al. (2022), ‘Magnetic control of tokamak plasmas through deep reinforcement learning’, *Nature* **602**(7897), 414–419.
- den Hengst, F., François-Lavet, V., Hoogendoorn, M. & van Harmelen, F. (2022), ‘Planning for potential: efficient safe reinforcement learning’, *Machine Learning* pp. 1–20.
- Dennis, M., Jaques, N., Vinitzky, E., Bayen, A., Russell, S., Critch, A. & Levine, S. (2020), ‘Emergent complexity and zero-shot transfer via unsupervised environment design’, *Advances in neural information processing systems* **33**, 13049–13061.
- Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018), ‘Bert: Pre-training of deep bidirectional transformers for language understanding’, *arXiv preprint arXiv:1810.04805* .
- Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y. & Zhokhov, P. (2017), ‘Openai baselines’.

- Duchi, J., Hazan, E. & Singer, Y. (2011), ‘Adaptive subgradient methods for online learning and stochastic optimization.’, *Journal of machine learning research* **12**(7).
- Efroni, Y., Mannor, S. & Pirota, M. (2020), ‘Exploration-exploitation in constrained mdps’, *arXiv e-prints* pp. arXiv–2003.
- El Mqirmi, P., Belardinelli, F. & León, B. G. (2021), An abstraction-based method to check multi-agent deep reinforcement-learning behaviors, in ‘Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems’, pp. 474–482.
- Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S. & Kavukcuoglu, K. (2018), IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures, in J. G. Dy & A. Krause, eds, ‘Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018’, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 1406–1415.
- Eysenbach, B., Gupta, A., Ibarz, J. & Levine, S. (2018), Diversity is All You Need: Learning Skills without a Reward Function.
URL: <https://openreview.net/forum?id=SJx63jRqFm>
- Faldor, M., Chalumeau, F., Flageat, M. & Cully, A. (2023), Map-elites with descriptor-conditioned gradients and archive distillation into a single policy, in S. Silva & L. Paquete, eds, ‘Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023’, ACM, pp. 138–146.
URL: <https://doi.org/10.1145/3583131.3590503>
- Fawzi, A., Balog, M., Huang, A., Hubert, T., Romera-Paredes, B., Barekatin, M., Novikov, A., Ruiz, F. J., Schrittwieser, J., Swirszcz, G. et al. (2022), ‘Discovering faster matrix multiplication algorithms with reinforcement learning’, *Nature* **610**(7930), 47–53.
- Flageat, M., Lim, B. & Cully, A. (2023), ‘Multiple hands make light work: Enhancing quality and diversity using map-elites with multiple parallel evolution strategies’, *CoRR* **abs/2303.06137**.
URL: <https://doi.org/10.48550/arXiv.2303.06137>

- Fodor, J. A. & Pylyshyn, Z. W. (1988), 'Connectionism and cognitive architecture: A critical analysis', *Cognition* **28**(1-2), 3–71.
- Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P. & Whiteson, S. (2017), Stabilising experience replay for deep multi-agent reinforcement learning, in 'Proceedings of the 34th International Conference on Machine Learning-Volume 70', JMLR. org, pp. 1146–1155.
- Fulton, N. & Platzer, A. (2018), Safe reinforcement learning via formal methods: Toward safe control through proof and learning, in 'Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Louisiana, USA.', AAAI Press.
- Fulton, N. & Platzer, A. (2019), Verifiably safe off-model reinforcement learning, in T. Vojnar & L. Zhang, eds, 'Tools and Algorithms for the Construction and Analysis of Systems', Springer International Publishing, Cham, pp. 413–430.
- Furelos-Blanco, D., Law, M., Jonsson, A., Broda, K. & Russo, A. (2021), 'Induction and exploitation of subgoal automata for reinforcement learning', *Journal of Artificial Intelligence Research* **70**, 1031–1116.
- G. León, B., Riccio, F., Subramanian, K., Wurman, P. R. & Stone, P. (2024), 'Discovering creative behaviours through duplex: Diverse universal features for policy exploration', *Under Review*.
- Gao, S. (2020), 'Hephaestus and talos', *Singapore Journal of Legal Studies* pp. 143–164.
- Garcia, J. & Fernández, F. (2015a), 'A comprehensive survey on safe reinforcement learning', *Journal of Machine Learning Research* **16**(1), 1437–1480.
- Garcia, J. & Fernández, F. (2015b), 'A comprehensive survey on safe reinforcement learning', *Journal of Machine Learning Research* **16**(1), 1437–1480.
- Gehring, J., Synnaeve, G., Krause, A. & Usunier, N. (2021), Hierarchical Skills for Efficient Exploration.
URL: <https://openreview.net/forum?id=NbaEmFm2mUW>
- Gelbin, C. S. (2021), 'The golem: From enlightenment monster to artificial intelligence', *Bulletin of the German Historical Institute* **69**(2022), 79–94.

- Gers, F. A., Schmidhuber, J. & Cummins, F. (2000), ‘Learning to forget: Continual prediction with lstm’, *Neural computation* **12**(10), 2451–2471.
- Ghahramani, Z. (2003), Unsupervised learning, in ‘Summer school on machine learning’, Springer, pp. 72–112.
- Giacobbe, M., Hasanbeig, M., Kroening, D. & Wijk, H. (2021a), ‘Shielding atari games with bounded prescience’.
- Giacobbe, M., Hasanbeig, M., Kroening, D. & Wijk, H. (2021b), Shielding atari games with bounded prescience, in ‘Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems’, p. 1507–1509.
- Giacomo, G. D., Iocchi, L., Favorito, M. & Patrizi, F. (2019), Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications, in J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith & S. Srivastava, eds, ‘Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019’, AAAI Press, pp. 128–136.
URL: <https://aaai.org/ojs/index.php/ICAPS/article/view/3549>
- Glorot, X., Bordes, A. & Bengio, Y. (2011), Deep sparse rectifier neural networks, in ‘Proceedings of the fourteenth international conference on artificial intelligence and statistics’, pp. 315–323.
- Goodall, A. W. & Belardinelli, F. (2023), Approximate model-based shielding for safe reinforcement learning, in ‘ECAI 2023’, IOS Press, pp. 883–890.
- Goodfellow, I., Bengio, Y., Courville, A. & Bengio, Y. (2016), *Deep learning*, Vol. 1, MIT press Cambridge.
- Grillotti, L. & Cully, A. (2022), Relevance-guided unsupervised discovery of abilities with quality-diversity algorithms, in J. E. Fieldsend & M. Wagner, eds, ‘GECCO ’22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022’, ACM, pp. 77–85.
URL: <https://doi.org/10.1145/3512290.3528837>

- Grillotti, L., Faldor, M., León, B. G. & Cully, A. (2023), Skill-conditioned policy optimization with successor features representations, *in* ‘Second Agent Learning in Open-Endedness Workshop’.
- Gu, A. & Dao, T. (2023), ‘Mamba: Linear-time sequence modeling with selective state spaces’, *arXiv preprint arXiv:2312.00752*.
- Gu, A., Goel, K. & Ré, C. (2022), Efficiently modeling long sequences with structured state spaces, *in* ‘The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022’, OpenReview.net.
URL: <https://openreview.net/forum?id=uYLFoz1vlAC>
- Ha, D. & Schmidhuber, J. (2018), ‘Recurrent world models facilitate policy evolution’, *Advances in neural information processing systems* **31**.
- Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S. (2018), Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, *in* J. G. Dy & A. Krause, eds, ‘Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018’, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 1856–1865.
URL: <http://proceedings.mlr.press/v80/haarnoja18b.html>
- Hafner, D., Lillicrap, T., Ba, J. & Norouzi, M. (2020), Dream to control: Learning behaviors by latent imagination, *in* ‘International Conference on Learning Representations’.
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H. & Davidson, J. (2019), Learning latent dynamics for planning from pixels, *in* ‘International conference on machine learning’, PMLR, pp. 2555–2565.
- Hafner, D., Lillicrap, T. P., Norouzi, M. & Ba, J. (2021), Mastering atari with discrete world models, *in* ‘International Conference on Learning Representations’.
- Hafner, D., Pasukonis, J., Ba, J. & Lillicrap, T. P. (2023), ‘Mastering diverse domains through world models’, *CoRR* **abs/2301.04104**.
URL: <https://doi.org/10.48550/arXiv.2301.04104>

- Hallak, A., Castro, D. D. & Mannor, S. (2015), ‘Contextual markov decision processes’, *CoRR abs/1502.02259*.
URL: <http://arxiv.org/abs/1502.02259>
- Hallak, A., Di Castro, D. & Mannor, S. (2015), ‘Contextual markov decision processes’, *arXiv preprint arXiv:1502.02259*.
- Hammond, L., Abate, A., Gutierrez, J. & Wooldridge, M. (2021), Multi-agent reinforcement learning with temporal logic specifications, in ‘Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems’, pp. 583–592.
- Hasanbeig, M., Jeppu, N. Y., Abate, A., Melham, T. & Kroening, D. (2021), Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning, in ‘The Thirty-Fifth {AAAI} Conference on Artificial Intelligence,{AAAI}’, Vol. 2, p. 36.
- Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Tibshirani, R. & Friedman, J. (2009), ‘Overview of supervised learning’, *The elements of statistical learning: Data mining, inference, and prediction* pp. 9–41.
- Hatanaka, W., Yamashina, R. & Matsubara, T. (2023), ‘Reinforcement learning of action and query policies with ltl instructions under uncertain event detector’, *IEEE Robotics and Automation Letters*.
- He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in ‘Proceedings of the IEEE international conference on computer vision’, pp. 1026–1034.
- He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, in ‘2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016’, IEEE Computer Society, pp. 770–778.
- He, P., Leon, B. G. & Belardinelli, F. (2022), Do androids dream of electric fences? safety-aware reinforcement learning with latent shielding, in ‘Proceedings of the Workshop on Artificial Intelligence Safety 2022 (SafeAI 2022)’, ceur-ws.org.

- Hill, F., Lampinen, A., Schneider, R., Clark, S., Botvinick, M., McClelland, J. L. & Santoro, A. (2020), Environmental drivers of systematicity and generalization in a situated agent, *in* ‘International Conference on Learning Representations’.
- Hill, F., Tieleman, O., von Glehn, T., Wong, N., Merzic, H. & Clark, S. (2021), Grounded language learning fast and slow, *in* ‘International Conference on Learning Representations’.
- Hochreiter, S. & Schmidhuber, J. (1997), ‘Long short-term memory’, *Neural Comput.* **9**(8), 1735–1780.
- Hsu, K.-C., Ren, A. Z., Nguyen, D. P., Majumdar, A. & Fisac, J. F. (2023), ‘Sim-to-lab-to-real: Safe reinforcement learning with shielding and generalization guarantees’, *Artificial Intelligence* **314**, 103811.
- Hu, J., Wellman, M. P. et al. (1998), Multiagent reinforcement learning: theoretical framework and an algorithm., *in* ‘ICML’, Vol. 98, Citeseer, pp. 242–250.
- Huang, W., Mordatch, I. & Pathak, D. (2020), One policy to control them all: Shared modular policies for agent-agnostic control, *in* ‘International Conference on Machine Learning’, PMLR, pp. 4455–4464.
- Huth, M. & Ryan, M. (2004), *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge university press.
- Hüttenrauch, M., Šošić, A. & Neumann, G. (2017), ‘Guided deep reinforcement learning for swarm systems’, *arXiv preprint arXiv:1709.06011* .
- Icarte, R. T., Klassen, T., Valenzano, R. & McIlraith, S. (2018), Using reward machines for high-level task specification and decomposition in reinforcement learning, *in* ‘International Conference on Machine Learning’, pp. 2112–2121.
- Icarte, R. T., Waldie, E., Klassen, T., Valenzano, R., Castro, M. & McIlraith, S. (2019), Learning reward machines for partially observable reinforcement learning, *in* ‘Advances in Neural Information Processing Systems’, pp. 15497–15508.

- Janner, M., Fu, J., Zhang, M. & Levine, S. (2019), ‘When to trust your model: Model-based policy optimization’, *Advances in neural information processing systems* **32**.
- Jiang, M., Grefenstette, E. & Rocktäschel, T. (2021), Prioritized level replay, in ‘International Conference on Machine Learning’, PMLR, pp. 4940–4950.
- Jiang, Y., Kolter, J. Z. & Raileanu, R. (2024), ‘On the importance of exploration for generalization in reinforcement learning’, *Advances in Neural Information Processing Systems* **36**.
- Jothimurugan, K., Alur, R. & Bastani, O. (2019), A composable specification language for reinforcement learning tasks, in H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. B. Fox & R. Garnett, eds, ‘Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada’, pp. 13021–13030.
- URL:** <https://proceedings.neurips.cc/paper/2019/hash/f5aa4bd09c07d8b2f65bad6c7cd3358f-Abstract.html>
- Jothimurugan, K., Bansal, S., Bastani, O. & Alur, R. (2021), ‘Compositional reinforcement learning from logical specifications’, *Advances in Neural Information Processing Systems* **34**.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A. et al. (2021), ‘Highly accurate protein structure prediction with alphafold’, *Nature* **596**(7873), 583–589.
- Junges, S., Jansen, N., Dehnert, C., Topcu, U. & Katoen, J.-P. (2016), Safety-constrained reinforcement learning for mdps, in ‘International conference on tools and algorithms for the construction and analysis of systems’, Springer, pp. 130–146.
- Karkus, P., Mirza, M., Guez, A., Jaegle, A., Lillicrap, T., Buesing, L., Heess, N. & Weber, T. (2020), ‘Beyond tabula-rasa: a modular reinforcement learning approach for physically embedded 3d sokoban’, *arXiv preprint arXiv:2010.01298*.
- Kemp, C. & Tenenbaum, J. B. (2008), ‘The discovery of structural form’, *Proc. Natl. Acad. Sci. USA* **105**(31), 10687–10692.

- Kiefer, J. & Wolfowitz, J. (1952), ‘Stochastic estimation of the maximum of a regression function’, *The Annals of Mathematical Statistics* pp. 462–466.
- Kingma, D. P. & Ba, J. (2014), ‘Adam: A method for stochastic optimization’, *arXiv preprint arXiv:1412.6980*.
- Kingma, D. P. & Ba, J. (2015), Adam: A method for stochastic optimization, in Y. Bengio & Y. LeCun, eds, ‘3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings’.
- URL:** <http://arxiv.org/abs/1412.6980>
- Kingma, D. P. & Welling, M. (2014), Auto-encoding variational bayes, in Y. Bengio & Y. LeCun, eds, ‘2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings’.
- URL:** <http://arxiv.org/abs/1312.6114>
- Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S. & Pérez, P. (2021), ‘Deep reinforcement learning for autonomous driving: A survey’, *IEEE Transactions on Intelligent Transportation Systems* **23**(6), 4909–4926.
- Kirk, R., Zhang, A., Grefenstette, E. & Rocktäschel, T. (2023), ‘A survey of zero-shot generalisation in deep reinforcement learning’, *Journal of Artificial Intelligence Research* **76**, 201–264.
- Kotovsky, K., Hayes, J. R. & Simon, H. A. (1985), ‘Why are some problems hard? evidence from tower of hanoi’, *Cognitive psychology* **17**(2), 248–294.
- Kramer, M. A. (1991), ‘Nonlinear principal component analysis using autoassociative neural networks’, *AIChE journal* **37**(2), 233–243.
- Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), ‘Imagenet classification with deep convolutional neural networks’, *Advances in neural information processing systems* **25**.
- Kumar, S., Kumar, A., Levine, S. & Finn, C. (2020), One solution is not all you need: Few-shot extrapolation via structured maxent RL, in H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan & H. Lin, eds, ‘Advances in Neural Information Processing Systems 33: Annual Conference on

Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual’.

URL: <https://proceedings.neurips.cc/paper/2020/hash/5d151d1059a6281335a10732fc49620e-Abstract.html>

Kuo, Y., Katz, B. & Barbu, A. (2020), ‘Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas’, *CoRR* **abs/2006.01110**.

Kupferman, O. & Vardi, M. Y. (2001), ‘Model checking of safety properties’, *Formal Methods in System Design* **19**(3), 291–314.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A. & Abbeel, P. (2018), Model-ensemble trust-region policy optimization, in ‘International Conference on Learning Representations’.

Küttler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatichi, M., Grefenstette, E. & Rocktäschel, T. (2020), The nethack learning environment, in H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan & H. Lin, eds, ‘Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual’.

URL: <https://proceedings.neurips.cc/paper/2020/hash/569ff987c643b4bedf504efda8f786c2-Abstract.html>

Lacerda, B., Parker, D. & Hawes, N. (2015), Optimal policy generation for partially satisfiable co-safe ltl specifications, in ‘Twenty-Fourth International Joint Conference on Artificial Intelligence’.

Lake, B. M. (2019), Compositional generalization through meta sequence-to-sequence learning, in ‘Advances in Neural Information Processing Systems’, pp. 9788–9798.

Lake, B. M. & Baroni, M. (2023), ‘Human-like systematic generalization through a meta-learning neural network’, *Nature* **623**(7985), 115–121.

Lampinen, A., Chan, S., Banino, A. & Hill, F. (2021), ‘Towards mental time travel: a hierarchical memory for reinforcement learning agents’, *Advances in Neural Information Processing Systems* **34**, 28182–28195.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), ‘Deep learning’, *nature* **521**(7553), 436.

- LeCun, Y., Bengio, Y. et al. (1995), 'Convolutional networks for images, speech, and time series', *The handbook of brain theory and neural networks* **3361**(10), 1995.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation applied to handwritten zip code recognition', *Neural computation* **1**(4), 541–551.
- Lehman, J. & Stanley, K. O. (2011), Evolving a diversity of virtual creatures through novelty search and local competition, in 'Proceedings of the 13th annual conference on Genetic and evolutionary computation', GECCO '11, Association for Computing Machinery, New York, NY, USA, pp. 211–218.
URL: <https://dl.acm.org/doi/10.1145/2001576.2001606>
- León, B. G. & Belardinelli, F. (2020), Extended markov games to learn multiple tasks in multi-agent reinforcement learning, in G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín & J. Lang, eds, 'ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)', Vol. 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, pp. 139–146.
URL: <https://doi.org/10.3233/FAIA200086>
- León, B. G., Shanahan, M. & Belardinelli, F. (2020), 'Systematic generalisation through task temporal logic and deep reinforcement learning', *arXiv preprint arXiv:2006.08767*.
- León, B. G., Shanahan, M. & Belardinelli, F. (2022a), Agent, do you see it now? systematic generalisation in deep reinforcement learning, in 'ICLR Workshop on Agent Learning in Open-Endedness'.
- León, B. G., Shanahan, M. & Belardinelli, F. (2022b), In a nutshell, the human asked for this: Latent goals for following temporal specifications, in 'The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022', OpenReview.net.
URL: <https://openreview.net/forum?id=rUwm9wCjURV>

- León, B. G., Shanahan, M. & Belardinelli, F. (2022c), In a nutshell, the human asked for this: Latent goals for following temporal specifications, *in* ‘The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022’, OpenReview.net.
URL: <https://openreview.net/forum?id=rUwm9wCjURV>
- Linnainmaa, S. (1970), The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors, PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki.
- Littman, M. L., Topcu, U., Fu, J., Isbell, C., Wen, M. & MacGlashan, J. (2017), ‘Environment-independent task specifications via gltl’, *arXiv preprint arXiv:1704.04341* .
- Liu, A., Xu, H., Van den Broeck, G. & Liang, Y. (2023), Out-of-distribution generalization by neural-symbolic joint training, *in* ‘Proceedings of the AAAI Conference on Artificial Intelligence’, Vol. 37, pp. 12252–12259.
- Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M. J. et al. (2021), ‘Algorithms for verifying deep neural networks’, *Foundations and Trends® in Optimization* **4**(3-4), 244–404.
- Ma, C., Ashley, D. R., Wen, J. & Bengio, Y. (2020), ‘Universal successor features for transfer reinforcement learning’, *CoRR* **abs/2001.04025**.
URL: <https://arxiv.org/abs/2001.04025>
- Maas, A. L., Hannun, A. Y., Ng, A. Y. et al. (2013), Rectifier nonlinearities improve neural network acoustic models, *in* ‘Proc. icml’, Vol. 30, Atlanta, GA, p. 3.
- Mahajan, A., Rashid, T., Samvelyan, M. & Whiteson, S. (2019), Maven: Multi-agent variational exploration, *in* ‘Advances in Neural Information Processing Systems’, pp. 7611–7622.
- Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B. & Wu, J. (2019), The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision, *in* ‘7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019’, OpenReview.net.

- Marchesini, E., Corsi, D. & Farinelli, A. (2022), Exploring safer behaviors for deep reinforcement learning, in ‘Proceedings of the AAAI Conference on Artificial Intelligence’, Vol. 36, pp. 7701–7709.
- Marvin, M. & Seymour, A. P. (1969), ‘Perceptrons’, *Cambridge, MA: MIT Press* **6**, 318–362.
- Mason, G. R., Calinescu, R. C., Kudenko, D. & Banks, A. (2017), Assured reinforcement learning with formally verified abstract policies, in ‘9th International Conference on Agents and Artificial Intelligence (ICAART)’, York.
- McClelland, J. L., Botvinick, M. M., Noelle, D. C., Plaut, D. C., Rogers, T. T., Seidenberg, M. S. & Smith, L. B. (2010), ‘Letting structure emerge: connectionist and dynamical systems approaches to cognition’, *Trends in cognitive sciences* **14**(8), 348–356.
- Mittal, S., Lamb, A., Goyal, A., Voleti, V., Shanahan, M., Lajoie, G., Mozer, M. & Bengio, Y. (2020), Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules, in ‘Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event’, Vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 6972–6986.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. & Kavukcuoglu, K. (2016), Asynchronous methods for deep reinforcement learning, in ‘International conference on machine learning’, pp. 1928–1937.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013), ‘Playing atari with deep reinforcement learning’, *arXiv preprint arXiv:1312.5602* .
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015), ‘Human-level control through deep reinforcement learning’, *Nature* **518**(7540), 529.
- Mouret, J. & Clune, J. (2015), ‘Illuminating search spaces by mapping elites’, *CoRR* **abs/1504.04909**.
URL: <http://arxiv.org/abs/1504.04909>

- Mozifian, M., Fox, D., Meger, D., Ramos, F. & Garg, A. (2021), ‘Generalizing successor features to continuous domains for multi-task learning’.
- Muniraj, D., Vamvoudakis, K. G. & Farhood, M. (2018), Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep q-learning approach, *in* ‘2018 IEEE Conference on Decision and Control (CDC)’, IEEE, pp. 4141–4146.
- Neary, C., Xu, Z., Wu, B. & Topcu, U. (2021), Reward machines for cooperative multi-agent reinforcement learning, *in* F. Dignum, A. Lomuscio, U. Endriss & A. Nowé, eds, ‘AAMAS ’21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021’, ACM, pp. 934–942.
URL: <https://www.ifaamas.org/Proceedings/aamas2021/pdfs/p934.pdf>
- Nilsson, O. & Cully, A. (2021), Policy gradient assisted map-elites, *in* F. Chicano & K. Krawiec, eds, ‘GECCO ’21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021’, ACM, pp. 866–875.
URL: <https://doi.org/10.1145/3449639.3459304>
- Nix, D. A. & Weigend, A. S. (1994), Estimating the mean and variance of the target probability distribution, *in* ‘Proceedings of 1994 IEEE international conference on neural networks (ICNN’94)’, Vol. 1, IEEE, pp. 55–60.
- Odriozola-Olalde, H., Zamalloa, M. & Arana-Arexolaleiba, N. (2023), Shielded reinforcement learning: A review of reactive methods for safe learning, *in* ‘2023 IEEE/SICE International Symposium on System Integration (SII)’, IEEE, pp. 1–8.
- Oh, J., Singh, S., Lee, H. & Kohli, P. (2017), Zero-shot task generalization with multi-task deep reinforcement learning, *in* ‘Proceedings of the 34th International Conference on Machine Learning-Volume 70’, JMLR. org, pp. 2661–2670.
- Oliehoek, F. A., Amato, C. et al. (2016), *A concise introduction to decentralized POMDPs*, Vol. 1, Springer.
- OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V.,

Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, J. H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V. H., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M. B., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle, P., Turley,

N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W. & Zoph, B. (2024), ‘Gpt-4 technical report’.

URL: <https://arxiv.org/abs/2303.08774>

O’reilly, R. C. (2001), ‘Generalization in interactive networks: The benefits of inhibitory competition and hebbian learning’, *Neural computation* **13**(6), 1199–1241.

O’Reilly, R. C., Petrov, A. A., Cohen, J. D., Lebiere, C. J., Herd, S. A., Kriete, T., Calvo, I. P. & Symons, J. (2013), ‘How limited systematicity emerges: A computational cognitive neuroscience approach’, *The architecture of cognition: Rethinking fodor and Pylyshyn’s systematicity challenge*.

Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S. et al. (2020), Stabilizing transformers for reinforcement learning, in ‘International conference on machine learning’, PMLR, pp. 7487–7498.

Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E. & Rocktäschel, T. (2022), Evolving curricula with regret-based environment design, in ‘International Conference on Machine Learning’, PMLR, pp. 17473–17498.

Peters, J. & Schaal, S. (2008), ‘Reinforcement learning of motor skills with policy gradients’, *Neural Networks* **21**(4), 682–697.

URL: <https://doi.org/10.1016/j.neunet.2008.02.003>

Pierrot, T., Macé, V., Chalumeau, F., Flajolet, A., Cideron, G., Beguir, K., Cully, A., Sigaud, O. & Perrin-Gilbert, N. (2022), Diversity policy gradient for sample efficient quality-diversity optimization, in J. E. Fieldsend & M. Wagner, eds, ‘GECCO ’22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022’, ACM, pp. 1075–1083.

URL: <https://doi.org/10.1145/3512290.3528845>

- Pnueli, A. (1977), The temporal logic of programs, *in* '18th Annual Symposium on Foundations of Computer Science (sfcs 1977)', IEEE, pp. 46–57.
- Polyak, B. T. (1964), 'Some methods of speeding up the convergence of iteration methods', *Ussr computational mathematics and mathematical physics* **4**(5), 1–17.
- Pugh, J. K., Soros, L. B. & Stanley, K. O. (2016), 'Quality diversity: A new frontier for evolutionary computation', *Frontiers Robotics AI* **3**, 40.
URL: <https://doi.org/10.3389/frobt.2016.00040>
- Qiu, W., Mao, W. & Zhu, H. (2024), 'Instructing goal-conditioned reinforcement learning agents with temporal logic objectives', *Advances in Neural Information Processing Systems* **36**.
- Rabin, M. O. & Scott, D. (1959), 'Finite automata and their decision problems', *IBM journal of research and development* **3**(2), 114–125.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. et al. (2019), 'Language models are unsupervised multitask learners', *OpenAI blog* **1**(8), 9.
- Raileanu, R. & Fergus, R. (2021), Decoupling value and policy for generalization in reinforcement learning, *in* 'International Conference on Machine Learning', PMLR, pp. 8787–8798.
- Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J. & Whiteson, S. (2018), 'Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning', *arXiv preprint arXiv:1803.11485*.
- Ray, A., Achiam, J. & Amodei, D. (2019), 'Benchmarking safe exploration in deep reinforcement learning', *arXiv preprint arXiv:1910.01708* **7**(1), 2.
- Riskin, J. (2003), 'The defecating duck, or, the ambiguous origins of artificial life', *Critical inquiry* **29**(4), 599–633.
- Rockafellar, R. T. (1970), *Convex Analysis*, Princeton Landmarks in Mathematics and Physics, Princeton University Press.
- Rockafellar, R. T. (1993), 'Lagrange multipliers and optimality', *SIAM review* **35**(2), 183–238.

- Rosenblatt, F. (1958), ‘The perceptron: a probabilistic model for information storage and organization in the brain.’, *Psychological review* **65**(6), 386.
- Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), ‘Learning representations by back-propagating errors’, *nature* **323**(6088), 533–536.
- Samvelyan, M., Raparthy, S. C., Lupu, A., Hambro, E., Markosyan, A. H., Bhatt, M., Mao, Y., Jiang, M., Parker-Holder, J., Foerster, J. et al. (2024), ‘Rainbow teaming: Open-ended generation of diverse adversarial prompts’, *arXiv preprint arXiv:2402.16822* .
- Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J. & Whiteson, S. (2019), The starcraft multi-agent challenge, in ‘Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems’, International Foundation for Autonomous Agents and Multiagent Systems, pp. 2186–2188.
- Santoro, A., Faulkner, R., Raposo, D., Rae, J. W., Chrzanowski, M., Weber, T., Wierstra, D., Vinyals, O., Pascanu, R. & Lillicrap, T. P. (2018), Relational recurrent neural networks, in S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi & R. Garnett, eds, ‘Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada’, pp. 7310–7321.
- Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P. W. & Lillicrap, T. (2017), A simple neural network module for relational reasoning, in I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan & R. Garnett, eds, ‘Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA’, pp. 4967–4976.
- Schaul, T., Horgan, D., Gregor, K. & Silver, D. (2015), Universal value function approximators, in ‘International conference on machine learning’, PMLR, pp. 1312–1320.
- Schillinger, P., Bürger, M. & Dimarogonas, D. V. (2019), ‘Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning’, *The international journal of robotics research* .
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017), ‘Proximal policy optimization algorithms’, *arXiv preprint arXiv:1707.06347* .

- Schuster, M. & Paliwal, K. K. (1997), 'Bidirectional recurrent neural networks', *IEEE transactions on Signal Processing* **45**(11), 2673–2681.
- Shalev-Shwartz, S., Shammah, S. & Shashua, A. (2016), 'Safe, multi-agent, reinforcement learning for autonomous driving', *arXiv preprint arXiv:1610.03295*.
- Shanahan, M., McDonell, K. & Reynolds, L. (2023), 'Role play with large language models', *Nature* **623**(7987), 493–498.
- Shanahan, M., Nikiforou, K., Creswell, A., Kaplanis, C., Barrett, D. G. T. & Garnelo, M. (2020), An explicitly relational neural network architecture, in 'Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event', Vol. 119 of *Proceedings of Machine Learning Research*, PMLR, pp. 8593–8603.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016), 'Mastering the game of go with deep neural networks and tree search', *nature* **529**(7587), 484.
- Simão, T. D., Jansen, N. & Spaan, M. T. (2021), Always safe: Reinforcement learning without safety constraint violations during training, in 'Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems', pp. 1226–1235.
- Smolensky, P. (1988), 'On the proper treatment of connectionism', *Behavioral and brain sciences* **11**(1), 1–23.
- Sutton, R. S. (1990), Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, in 'Machine learning proceedings 1990', Elsevier, pp. 216–224.
- Sutton, R. S. & Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT press.
- Sutton, R. S., Barto, A. G. et al. (1998), *Introduction to reinforcement learning*, Vol. 2, MIT press Cambridge.
- Sutton, R. S., McAllester, D. A., Singh, S. & Mansour, Y. (1999), Policy gradient methods for reinforcement learning with function approximation, in S. A. Solla, T. K. Leen & K. Müller, eds,

‘Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]’, The MIT Press, pp. 1057–1063.

URL: <http://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation>

Sutton, R. S. & Singh, S. P. (1994), On step-size and bias in temporal-difference learning, in ‘Proceedings of the eighth Yale workshop on adaptive and learning systems’, Citeseer, pp. 91–96.

Szepesvári, C. (2020), ‘Constrained mdps and the reward hypothesis’, *Musings about machine learning and other things (blog)* .

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. & Vicente, R. (2015), ‘Multiagent cooperation and competition with deep reinforcement learning’, *arXiv preprint arXiv:1511.08779* .

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. & Vicente, R. (2017), ‘Multiagent cooperation and competition with deep reinforcement learning’, *PloS one* **12**(4), e0172395.

Tan, M. (1993), Multi-agent reinforcement learning: Independent vs. cooperative agents, in ‘Proceedings of the tenth international conference on machine learning’, pp. 330–337.

Tang, H., Hao, J., Lv, T., Chen, Y., Zhang, Z., Jia, H., Ren, C., Zheng, Y., Fan, C. & Wang, L. (2018), ‘Hierarchical deep multiagent reinforcement learning’, *arXiv preprint arXiv:1809.09332* .

Tesauro, G. (1994), ‘Td-gammon, a self-teaching backgammon program, achieves master-level play’, *Neural computation* **6**(2), 215–219.

Tesauro, G. et al. (1995), ‘Temporal difference learning and td-gammon’, *Communications of the ACM* **38**(3), 58–68.

Tieleman, T., Hinton, G. et al. (2012), ‘Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude’, *COURSERA: Neural networks for machine learning* **4**(2), 26–31.

Tjanaka, B., Fontaine, M. C., Togelius, J. & Nikolaidis, S. (2022), Approximating gradients for differentiable quality diversity in reinforcement learning, in J. E. Fieldsend & M. Wagner,

eds, ‘GECCO ’22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022’, ACM, pp. 1102–1111.

URL: <https://doi.org/10.1145/3512290.3528705>

Todorov, E., Erez, T. & Tassa, Y. (2012), Mujoco: A physics engine for model-based control, in ‘2012 IEEE/RSJ International Conference on Intelligent Robots and Systems’, IEEE, pp. 5026–5033.

Toro Icarte, R., Klassen, T. Q., Valenzano, R. & McIlraith, S. A. (2018), Teaching multiple tasks to an rl agent using ltl, in ‘Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems’, International Foundation for Autonomous Agents and Multiagent Systems, pp. 452–461.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S. & Scialom, T. (2023), ‘Llama 2: Open foundation and fine-tuned chat models’.

URL: <https://arxiv.org/abs/2307.09288>

Turing, A. M. (2009), *Computing machinery and intelligence*, Springer.

Uchibe, E. & Doya, K. (2007), Constrained reinforcement learning from intrinsic and extrinsic rewards, in ‘2007 IEEE 6th International Conference on Development and Learning’, IEEE, pp. 163–168.

Vaezipoor, P., Li, A., Icarte, R. T. & McIlraith, S. (2021), ‘Ltl2action: Generalizing ltl instructions for multi-task rl’, *International Conference of Machine Learning*.

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017), Attention is all you need, *in* ‘Advances in neural information processing systems’, pp. 5998–6008.
- Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., Dudzik, A., Huang, A., Georgiev, P., Powell, R. et al. (2019), ‘Alphastar: Mastering the real-time strategy game starcraft ii’, *DeepMind blog* p. 2.
- Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J. P., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J. & Tsing, R. (2017), ‘Starcraft II: A new challenge for reinforcement learning’, *CoRR* **abs/1708.04782**.
URL: <http://arxiv.org/abs/1708.04782>
- Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y. et al. (2024), ‘A survey on large language model based autonomous agents’, *Frontiers of Computer Science* **18**(6), 186345.
- Wang, R., Lehman, J., Clune, J. & Stanley, K. O. (2019), ‘Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions’, *arXiv preprint arXiv:1901.01753*.
- Wen, M., Papusha, I. & Topcu, U. (2017), Learning from demonstrations with high-level side information, *in* ‘Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence’.
- Wen, M. & Topcu, U. (2016), Probably approximately correct learning in stochastic games with temporal logic specifications., *in* ‘IJCAI’, pp. 3630–3636.
- Williams, R. J. (1992), ‘Simple statistical gradient-following algorithms for connectionist reinforcement learning’, *Machine learning* **8**(3-4), 229–256.
- Wooldridge, M. (2009), *An introduction to multiagent systems*, John wiley & sons.

- Wurman, P. R., Barrett, S., Kawamoto, K., MacGlashan, J., Subramanian, K., Walsh, T. J., Capobianco, R., Devlic, A., Eckert, F., Fuchs, F. et al. (2022), ‘Outracing champion gran turismo drivers with deep reinforcement learning’, *Nature* **602**(7896), 223–228.
- Xu, Z., Gavran, I., Ahmad, Y., Majumdar, R., Neider, D., Topcu, U. & Wu, B. (2020), Joint inference of reward machines and policies for reinforcement learning, *in* ‘Proceedings of the International Conference on Automated Planning and Scheduling’, Vol. 30, pp. 590–598.
- Yang, Y., Jiang, Y., Liu, Y., Chen, J. & Li, S. E. (2023), ‘Model-free safe reinforcement learning through neural barrier certificate’, *IEEE Robotics and Automation Letters* **8**(3), 1295–1302.
- Ye, D., Zhang, M. & Yang, Y. (2015), ‘A multi-agent framework for packet routing in wireless sensor networks’, *sensors* **15**(5), 10026–10047.
- Yu, H., Zhang, H. & Xu, W. (2018), Interactive grounded language acquisition and generalization in a 2d world, *in* ‘6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings’, OpenReview.net.
- Yuan, L. Z., Hasanbeig, M., Abate, A. & Kroening, D. (2019), ‘Modular deep reinforcement learning with temporal logic specifications’, *arXiv preprint arXiv:1909.11591*.
- Zachrisson, L. E. (1964), *Markov games*, Vol. 52, Princeton University Press Princeton, NJ.
- Zahavy, T., O’Donoghue, B., Desjardins, G. & Singh, S. (2021), Reward is enough for convex mdps, *in* M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang & J. W. Vaughan, eds, ‘Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual’, pp. 25746–25759.
URL: <https://proceedings.neurips.cc/paper/2021/hash/d7e4cdde82a894b8f633e6d61a01ef15-Abstract.html>
- Zahavy, T., Schroecker, Y., Behbahani, F. M. P., Baumli, K., Flennerhag, S., Hou, S. & Singh, S. (2023), Discovering policies with domino: Diversity optimization maintaining near optimality, *in* ‘The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023’, OpenReview.net.
URL: <https://openreview.net/pdf?id=kjkdzBW3b8p>

- Zahavy, T., Veeriah, V., Hou, S., Waugh, K., Lai, M., Leurent, E., Tomasev, N., Schut, L., Hassabis, D. & Singh, S. (2024), ‘Diversifying AI: Towards Creative Chess with AlphaZero’. arXiv:2308.09175 [cs].
URL: <http://arxiv.org/abs/2308.09175>
- Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E. et al. (2018), Deep reinforcement learning with relational inductive biases, in ‘International Conference on Learning Representations’.
- Zelikman, E., Harik, G., Shao, Y., Jayasiri, V., Haber, N. & Goodman, N. D. (2024), ‘Quiet-star: Language models can teach themselves to think before speaking’, *arXiv preprint arXiv:2403.09629* .
- Zhang, K., Yang, Z. & Başar, T. (2021), ‘Multi-agent reinforcement learning: A selective overview of theories and algorithms’, *Handbook of reinforcement learning and control* pp. 321–384.
- Zhao, M., Liu, Z., Luan, S., Zhang, S., Precup, D. & Bengio, Y. (2021), A consciousness-inspired planning agent for model-based reinforcement learning, in ‘Advances in Neural Information Processing Systems’.
- Zhao, W., He, T., Chen, R., Wei, T. & Liu, C. (2023), ‘State-wise safe reinforcement learning: A survey’, *arXiv preprint arXiv:2302.03122* .
- Zhou, W. & Li, W. (2018), Safety-aware apprenticeship learning, in ‘International Conference on Computer Aided Verification’, Springer, pp. 662–680.

Appendix A

Appendix for Systematic Generalisation with Abstract Operators

A.1 Hyperparameters

The A2C algorithm uses a learning rate of $8e^{-5}$, this value for the learning rate is the one that give us the best results when doing grid search between $1e^{-5}$ and $1e^{-3}$, except for the ResNet that shows better results with the value of $2e^{-5}$ that we use in the experiments reported in the main section. The entropy and value loss regularisation terms are $1e^{-3}$ (best results we found in the range $[1e^{-4}, 0]$ and 0.5 (as in Mnih et al. (2016)) respectively. The expected-reward discount factor is 0.99, which is typical value for this term. Batch size is set to 512 values, which gives the best results in the range of $([128, 2048])$.

A.2 Hardware

The various architectures were trained on computing clusters with GPUs such as Nvidia Tesla K80, Testla T4 or TITAN Xp. We employed Intel Xeon CPUs and consumed 7 GB of RAM per 5 independent runs working in parallel. Each experiment typically takes 2 days to train on this hardware. All the results presented in the main text and supplementary material are obtained from the aforementioned five independent runs per experiment.

A.3 Translating TTL into LTL_f

It can be easily inferred that our task temporal logic (TTL) is a fragment of the more popular linear-time temporal logic over finite traces (LTL_f) (Baier & Katoen 2008). The syntax of LTL_f is defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2$$

Note that both TTL and LTL_f are defined over finite traces. Below we provide truth-preserving translations of TTL into LTL_f .

Definition A.1. *Translation τ from TTL to LTL_f is defined as follows:*

$$\begin{aligned} \tau(\alpha) &= \diamond p_\alpha, \text{ where } p_\alpha \text{ is the fulfilment} \\ &\quad \text{condition associated with task } \alpha \end{aligned}$$

$$\tau(\alpha \sim) = \diamond(\bigvee_{\beta \neq \alpha} p_\beta \wedge \neg p_\alpha)$$

$$\tau(\alpha \vee \alpha') = \tau(\alpha) \vee \tau(\alpha')$$

$$\tau(\phi \cup \phi') = \tau(\phi) \vee \tau(\phi')$$

$$\tau(\phi; \phi') = \begin{cases} \diamond(p_\alpha \wedge \tau(\phi)) & \text{if } \phi = \alpha \\ \diamond((\bigvee_{\beta \neq \alpha} p_\beta \wedge \neg p_\alpha) \wedge \tau(\phi)) & \text{if } \phi = \alpha \sim \\ \diamond((p_\alpha \vee p_{\alpha'}) \wedge \tau(\phi)) & \text{if } \phi = \alpha \vee \alpha' \\ \tau(\phi_1; (\phi_2; \phi')) & \text{if } \phi = \phi_1; \phi_2 \\ \tau((\phi_1; \phi') \cup (\phi_2; \phi')) & \text{if } \phi = \phi_1 \cup \phi_2 \end{cases}$$

In the main body of this work we stated that TTL is a fragment of LTL_f , and supported such claim by explaining that translations τ_1 and τ_2 preserve the interpretation of formulae in TTL. We include now a detailed proof for this claim:

Proposition 2. *Given a model $\mathcal{N}$ and trace λ , for every formula ϕ in TTL,*

$$(\mathcal{N}, \lambda) \models \phi \text{ iff } (\mathcal{N}, \lambda) \models \tau(\phi)$$

Proof. The proof is by induction on the structure of formula ϕ . The base case follows immediately by the semantics of TTL and LTL_f .

As for the induction step, the case of interest is for formulae of type $\phi; \phi'$. In particular, consider the case for $\phi = \alpha$. We then have that $(\mathcal{N}, \lambda) \models \alpha; \phi'$ iff for some $0 \leq j < |\lambda|$, $(\mathcal{N}, \lambda[0, j]) \models \alpha$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models \phi'$. By induction hypothesis, this is equivalent to $(\mathcal{N}, \lambda[0, j]) \models \diamond p_\alpha$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models \tau(\phi')$. Finally, this is equivalent to $(\mathcal{N}, \lambda) \models \diamond(p_\alpha \wedge \tau(\phi'))$. The other cases. are dealt with similarly.

Finally, the case for $\phi \cup \phi'$ follows by induction hypothesis and the distributivity of $\diamond$ over $\vee$. $\square$

Appendix B

Appendix for Network Architectures with Latent Goals

B.1 Symbolic Module

Here we present the symbolic module of Chapter 7, which is equivalent to the one we introduced in Chapter 6, adapted to work with SATTL.

Algorithm 7 provides the pseudocode of the SM. The first component is the *extractor* $\mathcal{E}$ (Algorithm 8), which transforms the complex formula T into a list $\mathcal{K}$ consisting of all the sequences of atomic tasks α that satisfy T . As stated in the main document, the SM have access to an internal labelling function $\mathcal{L}_I : Z \rightarrow 2^{AP}$, where $\mathcal{L}_I$ is the restriction of $\mathcal{L}$ to the observation space and maps the observation into the set of atoms (e.g, `reached_sword` is an atom). Intuitively the labelling function acts as an *event detector* that fires when the propositions in AP hold in the environment. The second functionality is the *progression function* $\mathcal{P}$ (Algorithm 9), which given the output of $\mathcal{L}_I$ and the current list $\mathcal{K}$, selects the next task α for the NM and updates $\mathcal{K}$. This $\mathcal{P}$ differs from the one presented in Chapter 6 in Line 15 of Algorithm 9, where we create disjunctions both before and after the Until operator that is particular from SATTL.

Algorithm 7 Symbolic Module (SM)

```

1: Input: Instruction  $T$ 
2: Generate the accepted sequences of tasks  $\mathcal{K} \leftarrow \mathcal{E}(T)$ 
3: Retrieve the current observation:  $z$ 
4: Get the true proposition:  $p \leftarrow \mathcal{L}_I(z)$ 
5: Get the first task:  $\alpha \leftarrow \mathcal{P}(\mathcal{K}, p)$ 
6: repeat
7:   Get the next action:  $a \leftarrow \text{Neural Module (NM)}(z, \alpha)$ 
8:   Execute  $a$ , which updates  $z$ 
9:   Get the new true proposition:  $p \leftarrow \mathcal{L}_I(z)$ 
10:  Provide the reward:  $\text{NM} \leftarrow R(p)$ 
11:  if  $p == p_\alpha$  then
12:    Update  $\mathcal{K}, \alpha \leftarrow \mathcal{P}(\mathcal{K}, p)$ 
13: until  $\mathcal{K} == \emptyset$ 

```

Once α is selected, the NM follows the given instruction until it is satisfied or until the time horizon is reached, while the role of the SM is to evaluate the fulfillment of the task based on the signals from $\mathcal{L}_I$. From the outputs of $\mathcal{L}_I$, the SM generates a reward function R , which provides the reward signal for the NM. Note that from the NM perspective, this signal is the standard reward signal of a POMDP – but in our case associated to the instruction–.

B.2 Additional details of empirical results

Tables B.1-B.4 provide the specifics values of the empirical results in a table format. From the training results, we observe that latent-goal configurations can improve the performance also the with training instructions and that this improvement becomes more noticeable as environments become larger in size, i.e., as we get further from the training distribution of environments. The fact that the effects of latent-goal networks are most noticeable with unseen instructions aligns with the conclusions drawn from the main document, where we stated that this architectures are specially conceived to strength agents' ability to generalise to new tasks.

Algorithm 8 Extractor function, obtains the list of all the possible sequences of tasks

```

1: Function  $\mathcal{E}$ , Input:  $T$ 
2: Initialize the list for the sequences of tasks  $\mathcal{K}$ 
3: for each atomic task  $\alpha \in T$  do
4:   if  $\alpha$  is atomic positive or negative then
5:     for all  $Seq \in \mathcal{K}$  do
6:        $Seq.append(\alpha)$ 
7:   else
8:     // There are non-deterministic choices
9:     Initialize choice list:  $CL$ 
10:     $LK \leftarrow length(\mathcal{K})$ 
11:    for all  $Seq \in \mathcal{K}$  do
12:      for all choice  $\in \alpha$  do
13:         $CL.append(choice)$ 
14:        Generate a clone per choice  $Seq' \leftarrow Seq$ 
15:         $\mathcal{K}.append(Seq')$ 
16:    Initialize counter  $c \leftarrow -1$ 
17:    for  $i$  in  $range(length(\mathcal{K}))$  do
18:      if  $i \% LK == 0$  then
19:         $c++ = 1$ 
20:      We append a different choice to each sequence cloned  $\mathcal{K}[i].append(CL[c])$ 
return  $\mathcal{K}$ 

```

B.2.1 Statistical Analysis

To confirm that the standard error of the different conditions do not overlap each other, we perform a two-way analysis of variance (two-way ANOVA) with the independent variables being the neural networks used in the central modules and the architecture configuration, i.e., latent goal (ours) or standard configuration. The 2-way ANOVA is done across the maps of different size of Minecraft and MiniGrid respectively. Note that to evaluate the different hypotheses, we need to check whether the F value is greater than its corresponding F-critic or not, and to confirm that the P-value is smaller than Alpha.

Table B.5 shows the results of the ANOVA analysis with the experiments in Minecraft. They confirm that both the architecture configuration and the neural network have a significant impact of the performance of the agent. Noticeably, the ANOVA results show that there is not a significant interaction between the two variables, meaning that the neural network is not relevant in the general improvement in performance that latent goals grant in this setting. Still, note that this is analysis of the statistical impact averaged across all the networks. As discussed in Section 7.3.3 the latent goals do have a

Algorithm 9 Progression function, returns the next task to be solved.

```

1: Function  $\mathcal{P}$ , Input:  $\mathcal{K}, p_\alpha$ 
2: if  $p_\alpha \neq \emptyset$  then
3:   for all  $Seq \in \mathcal{K}$  do
4:     if  $p_\alpha$  fulfills  $Seq[0]$  then
5:        $Seq.pop(Seq[0])$ 
6:     else
7:        $\mathcal{K}.pop(Seq)$ 
8:  $\alpha' \leftarrow \emptyset$ 
9: if  $\mathcal{K} == \emptyset$  then return  $\alpha'$ 
10: else
11:   Select the head of the first sequence:  $\alpha' \equiv l \cup l' \leftarrow \mathcal{K}[0][0]$ 
12:   for all  $Seq \in \mathcal{K}$  do
13:     // look for a non-deterministic choice
14:     if  $\alpha'$  &  $Seq[0]$  can be combined then
15:        $\alpha' \leftarrow (l \vee l_0) \cup (l' \vee l'_0)$ , where  $Seq[0] \equiv l_0 \cup l'_0$ 
return  $\alpha'$ 

```

different impact with the PrediNet than with other networks.

Table B.6 shows the results of the ANOVA test with MiniGrid. Here, we see that neural networks have a significant impact in the training performance of the agent but that is not the case with the architecture, i.e., using latent goals does not have a significant impact in the global training performance. Nevertheless, we see that this changes with unseen instructions (test) where the latent goals have a stronger impact in performance than the neural network, being both statistically relevant. Last, from the interactions' results we see that both in training and test the interaction of the two variables have a significant impact in the final performance, i.e., the effect in performance that latent goals have is strongly dependant on the neural network and vice versa.

Regarding the differences in how network and architecture impact in Minecraft and MiniGrid, we believe that these are caused by the additional difficulty of MiniGrid observation and action settings. Specifically, MiniGrid only allows agents to move forward or turn around, and agents can only observe what is in front of them. This requires agents to rely more on their memory than they do in Minecraft, where they can observe the objects around them in any direction and have actions to move in four different directions. This point also motivated the need of using an schedule in the introduction of larger training maps in MiniGrid, as detailed in Section 7.3.1.

Table B.1: Results with either training (train) or OOD instructions (test) in 500 maps (per size) of different dimensions. Note that sizes 14 and 22 are OOD for all the agents. MC refers Minecraft whereas MG to MiniGrid. Results show the average reward and standard deviation from 10 independent runs (i.r.). Values are normalised so that 100 refers to the highest performance achieved by the best run globally in maps of the given size and benchmark. Best average rewards in each setting and size is bolded.

Map-size	MLP		RN		MHA		PrediNet	
	Train	Test	Train	Test	Train	Test	Train	Test
MG-7x7	18.2(5.7)	14.1(3.6)	20.6(5.4)	18.7 (6.4)	45.9 (26.6)	9.2(3.7)	19.1(12.2)	15.9 (7.3)
MC-7x7	33.4(7.9)	21.1(3.2)	5(1.9)	3.2 (1.6)	41.1(7.7)	36.6 (4.3)	21.8(3.7)	16.1(3.5)
MG-14x14	28.6(8.2)	21.1(5.7)	28.6(6.2)	27.5(6.0)	57.3 (17.8)	21.9 (4.7)	31.2(10.7)	25.6(7.3)
MC-14x14	38.7(6.5)	32.0(4.5)	8.3(2)	7.9(2.4)	51.7(9.3)	45.8 (7.6)	40.2(5)	31.9(6.2)
MG-22x22	27.8(8.5)	21.6(4.3)	26.9(10.1)	24.3(5.1)	54.9 (23.6)	22.7(6.8)	32.1(8.9)	24.1(8.0)
MC-22x22	47.8(3.6)	37.8(3.8)	9.2(1.5)	8.3(1.8)	56.9(10.7)	49.2(5)	48.7(10.3)	38(8.5)

Map-size	MLP ^{LG} (ours)		RN ^{LG} (ours)		MHA ^{LG} (ours)		PrediNet ^{LG} (ours)	
	Train	Test	Train	Test	Train	Test	Train	Test
MG-7x7	23.6(5.5)	21.6(2.6)	22.6(3.3)	21.8 (2.8)	42.0(24.9)	15.4(3.5)	32.1(9.3)	26.1 (5.8)
MC-7x7	40.2(5.7)	24.7(2.6)	4.4(1)	4.2 (0.7)	35.2(5)	31.6(5.3)	55.2 (17.6)	29.5(8.2)
MG-14x14	31.4(4.2)	24.6(3.8)	30.1(6.1)	29.2(6.9)	50.4(14.9)	24.0 (9.4)	37.0(6.7)	30.6 (5.5)
MC-14x14	41.7(5.1)	35.4(4.2)	13.1(2.6)	11.5(2.2)	48.1(4.4)	41.8 (4.9)	67 (5.2)	51.7 (6.5)
MG-22x22	31.0(6.3)	22.5(5.0)	34.2(2.3)	25.1(1.5)	53.3(21.4)	25.4(8.2)	39.9(2.7)	29.7 (7.0)
MC-22x22	52.6(8.6)	41.3(7.7)	11.7(1.5)	10.7(2.1)	58.4 (4.5)	51.5(5.5)	77.5 (9.1)	65.3 (6.1)

Table B.2: Results from 5 i.r. of a vanilla BRIM (BRIM), a BRIM with residual connection (ResBRIM) and BRIM with latent goals (BRIM^{LG}). Best results for each setting and size are in bold.

Map-size	BRIM		ResBRIM		BRIM ^{LG} (ours)		Random walker
	Train	Test	Train	Test	Train	Test	
MG-7x7	19.6(6.2)	15.1(4.7)	19.2(4.8)	16.7 (3.6)	23.8 (5.4)	17.1 (5.3)	4.3(0.4)
MC-7x7	27.9(6.3)	15.6(2.8)	58.6(10.4)	15.8 (2.3)	77.2 (19.5)	26.4 (7.0)	3.7(0.1)
MG-14x14	25.3(5.4)	19.9(6.2)	33.7(3.9)	25.0 (1.4)	38.8 (9.8)	26.8 (5.6)	9.5(1.3)
MC-14x14	37.9(8.2)	26.5(6.7)	63.5(5.5)	39.6(7.0)	77.3 (21.3)	46.8 (8.1)	11.5(1.4)
MG-22x22	26.5(7.1)	20.0 (7.6)	27.9 (3.0)	25.3(2.1)	48.5 (26.0)	30.3 (6.6)	13.6(4.2)
MC-22x22	41.5(8.4)	29.3(4.1)	60.6(10.4)	40.7(8.2)	79.8 (17.0)	58.4 (14.8)	11.1(0.5)

B.3 SATTL and LTL_f

We stated that SATTL is a fragment of the widely-used LTL_f . The syntax of LTL_f is defined as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \text{U} \varphi_2$$

Since both LTL_f and SATTL are defined over finite traces, we can directly introduce the following translations:

Definition B.1. Translation τ from Task Temporal Logic to LTL_f are defined as follows:

Table B.3: Ablation studies in Minecraft (5 i.r. per variant). PrediNet^{LG}_{noSub} and PrediNet^{LG}_{noPos} are variants of PrediNet^{LG} without the element-wise subtraction and the feature coordinates respectively. PNMHA^{LG} uses an MHA in CM2 while MHAPN^{LG} uses an MHA in CM1. Best results are in bold

Ablations	7x7		14x14		22x22	
PrediNet ^{LG} variants	Train	Test	Train	Test	Train	Test
PrediNet ^{LG} _{noSub}	41 (8)	22.8 (2.8)	64.7 (7.9)	42.3 (6.9)	63.7 (7.8)	46.4 (8.1)
PrediNet ^{LG} _{noPos}	50.6 (9.9)	29.3 (5)	68.2 (10.8)	48.1 (5.4)	70.6 (9.3)	56.4 (9.2)
PNMHA ^{LG}	39.3 (7.6)	29.2 (6.2)	52.8 (3.9)	46.5 (7.9)	64.6 (6.7)	45.2 (10)
MHAPN ^{LG}	49 (10)	36.6 (7.5)	59.9 (9.8)	45.6 (4.9)	57.4 (6.4)	51.6 (7.6)

Table B.4: Ablation studies with PrediNet^{LG} in Minecraft (5 i.r. per variant). In the first set we explore the impact of different bottleneck sizes. In the second we contrast having L_s task-agnostic or not. For every ablation set, best results are bolded.

Ablations	7x7		14x14		22x22	
Bottleneck size	Train	Test	Train	Test	Train	Test
4	57.0 (8.4)	26.0 (10.9)	72.0 (15.7)	60.4 (5.9)	74.8 (17.2)	58.3 (12.9)
16	51.2 (11.6)	25.3 (6.1)	81.7 (17.9)	49.4 (14.2)	86.1 (12.6)	61.8 (10.3)
64	49.8 (10.2)	19.1 (5.7)	69.3 (14.3)	35.2 (13.9)	64.2 (15.1)	38.1 (13.3)
256	56.7 (14.7)	25.4 (11.3)	70.5 (13.0)	33.2 (9.5)	71.7 (10.1)	41.2 (17.3)
L_s data type						
Task-agnostic	51.2 (11.6)	25.3 (5.4)	81.7 (17.9)	49.4 (12.6)	86.1 (12.6)	61.8 (10.3)
Not Task-agnostic	48.1 (13.6)	23.0 (4.6)	58.7 (10.4)	33.7 (5.8)	59.4 (14.1)	44.4 (7.3)

$$\begin{aligned}
\tau(\alpha) &= IUl' \\
\tau(T \cup T') &= \tau(T) \vee \tau(T') \\
\tau(T; T') &= \begin{cases} IU(l' \wedge \tau(T')) & \text{if } T = \alpha \\ \tau(T_1; (T_2; T')) & \text{if } T = T_1; T_2 \\ \tau((T_1; T') \cup (T_2; T')) & \text{if } T = T_1 \cup T_2 \end{cases}
\end{aligned}$$

We immediately prove that translation τ preserve the interpretation of formulae in SATTL.

Proposition 3. *Given a model $\mathcal{N}$ and trace λ , for every formula T in SATTL,*

$$(\mathcal{N}, \lambda) \models T \text{ iff } (\mathcal{N}, \lambda) \models \tau(T)$$

Proof. The proof is by induction on the structure of formula T . The base case follows immediately by the semantics of SATTL and LTL_f.

As for the induction step, the case of interest is for formulae of type $T; T'$. In particular, $(\mathcal{N}, \lambda) \models T; T'$

Table B.5: Two-way ANOVA test with Alpha= 0.05 in Minecraft. Neural network refers to the impact of the different neural networks in the central modules, whereas architecture configuration refers to the impact of whether using a latent goal or a standard configuration.

Train	<i>F</i>	<i>P – value</i>	<i>F – critic</i>
Architecture configuration	43.56	2.74exp–10	3.88
Neural Network	237.31	2.13exp–70	2.64
Interactions	31.27	5.16exp–17	2.64
Test	<i>F</i>	<i>P – value</i>	<i>F – critic</i>
Architecture configuration	23.63	2.15exp–6	3.88
Neural Network	166.75	1.26exp–57	2.64
Interactions	16.26	1.26exp–09	2.64

Table B.6: Two-way ANOVA test with Alpha = 0.05 in MiniGrid.

Train	<i>F</i>	<i>P – value</i>	<i>F – critic</i>
Architecture configuration	1.37	0.24	3.93
Neural Network	39.57	1.63exp–17	2.69
Interactions	1.86	0.14	2.69
Test	<i>F</i>	<i>P – value</i>	<i>F – critic</i>
Architecture configuration	8.57	0.004	3.93
Neural Network	5.51	0.001	2.69
Interactions	0.53	0.6	2.69

iff for some $0 \leq j < |\lambda|$, $(\mathcal{N}, \lambda[0, j]) \models T$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models T'$. By induction hypothesis, this is equivalent to $(\mathcal{N}, \lambda[0, j]) \models IUl'$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models \tau(T')$ in the case of $T = IUl'$. Finally, this is equivalent to $(\mathcal{N}, \lambda) \models IU(l' \wedge \tau(T'))$. The cases for $T = T_1; T_2$ and $T = T_1 \cup T_2$ are dealt with similarly.

Finally, the case for $T \cup T'$ follows by induction hypothesis and the distributivity of $\vee$. □

Appendix C

Appendix for Policy Exploration

C.1 Environment details

We demonstrated that DUPLEX effectively achieves diversity in multiple benchmarks. In the Mujoco environments diversity is measured through the agent’s observations, which corresponds to the angles and velocities of the different parts of the agent. In GT, diversity is measured through action intensities (break, throttle and steering), wheel angles, and distances to the track edges.

Note that these diversity features were chosen to be aligned with the objectives in each environment. Nevertheless, we can choose any feature of the observation to be optimised for diversity. Such an arbitrary selection of features intuitively leads to different optimisation objectives and thus learned strategies. For instance, in GranTurismo, we could select the frequency of hitting other cars, and this would lead to a population of policies that are all driving as fast as possible while exploring the spectrum of aggressiveness: ranging from a very timid behavior (letting opponents pass to avoid contact) to a very aggressive one (hitting cars almost intentionally if the collision does not result in a significant speed loss).

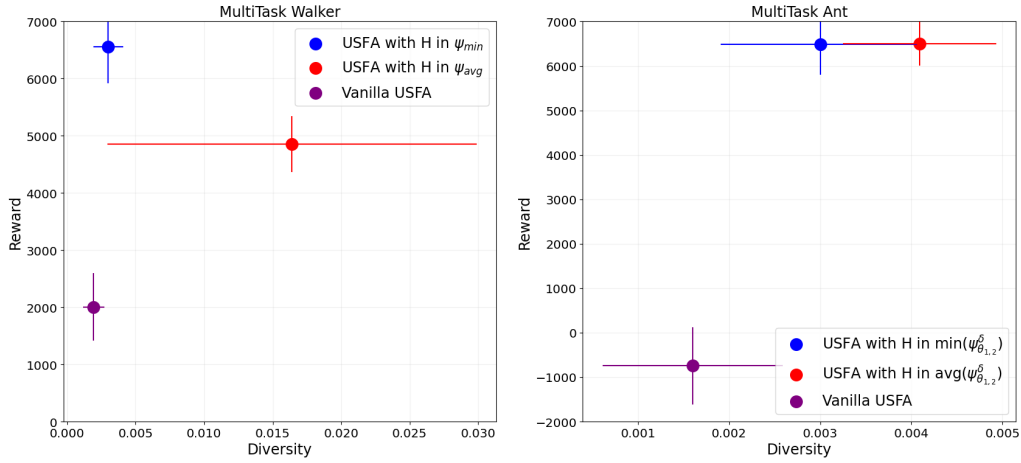


Figure C.1: Ablation on USFA, we see that incorporating entropy to the learning objective of the successor features estimator as in DUPLEX yields a significant boost to accumulate rewards.

C.2 Ablation in USFA

In this section we provide empirical evidence that incorporating entropy to the objective function in SFs can be beneficial to existing frameworks such as USFA.

Specifically, we investigate if updating USFA objective function with Equation 8.7 that we introduced for DUPLEX also yields a stronger performance than this well known framework. Figure C.1 illustrates our findings where we show that such an addition (i) enables effective learning in multi-task mujoco environments and (ii) significantly improves performance. Moreover, the ablation also shows that taking the average of the critic estimates is not beneficial in USFA. This provides further evidence that taking the average in DUPLEX is beneficial since the overestimation of the intrinsic rewards is caused by the accuracy of the distance between SFs not the magnitude of the SFs themselves, which is something not present in USFA.

C.3 Hyperparameters

Table C.1 lists the hyperparameters by DUPLEX and its neural networks. Our algorithm is designed to be calibrated through two hyperparameters ρ and β . The former regulates how much the user wants to weight the intrinsic rewards w.r.t. the extrinsic reward, while the latter defines the extrinsic rewards

lower bound which is used to better define the optimality region that the agent should explore when learning diverse policies.

Table C.1: Hyperparameters of DUPLEX.

HYPERPARAMETER	MULTI-TASK MUJOCO	GT
N	10	10
ρ	$[0.9, 0.7]$	$[0.88, 0.8]$
β	0.5	0.97
α_χ	$1e^{-5}$	$1e^{-5}$
ε_χ	20	20
k FROM SIGMOID	5	5
$\alpha_{\psi^{\text{avg}}}$	0.9999	0.9999
$\alpha_{v^{\text{avg}}}$	0.999	0.999
$\hat{v}$ SAMPLE RATE	0.3	0.3
κ LEARNING RATE	$1e^{-2}$	-
SAC/QRSAC LEARNING RATE	$3e^{-4}$	$2.5e^{-4}$
TARGET CRITIC TARGET SOFT UPDATE RATE	0.005	0.005
CLIP GRADIENT GLOBAL NORM	3.0	
OBS ENCODER SIZE	256	2048
CONTEXT ENCODER SIZE	64	-
POLICY ENCODER SIZE	256	256
TORSO SIZE	2x256	4x2048
CRITIC HEAD HIDDEN LAYER SIZE	256	2048
ENTROPY LEARNING RATE	$1e^{-4}$	FIXED ENTROPY

Appendix D

Appendix for Extended Markov Games

Here we prove Corollary 1 in Chapter 4. First, by next result every *co-safe LTL* formula is true iff it is satisfied by a finite sequence.

Proposition 4. *Let $\sigma \in (2^{\mathcal{AP}})^\omega$ be an infinite sequence, and φ a co-safe LTL formula. Then,*

$$\sigma \models \varphi \text{ iff for some } i \geq 0, \sigma_{\leq i} \models \varphi \quad (\text{D.1})$$

Proof. We first prove direction $\Rightarrow$ by induction on the structure of φ . Suppose $\sigma \models \varphi$. We compute the value i by induction on the structure of φ as follows:

$$\begin{aligned} i(\sigma, p) &= 0 = i(\sigma, \neg p) \\ i(\sigma, \varphi_1 \wedge \varphi_2) &= \max\{i(\sigma, \varphi_1), i(\sigma, \varphi_2)\} \\ i(\sigma, \varphi_1 \vee \varphi_2) &= \min\{i(\sigma, \varphi_1), i(\sigma, \varphi_2)\} \\ i(\sigma, \bigcirc \varphi_1) &= i(\sigma_{\geq 1}, \varphi_1) + 1 \\ i(\sigma, \varphi_1 \text{ U } \varphi_2) &= \max\{\{i(\sigma_{\geq k}, \varphi_1)\}_{k < j}, i(\sigma_{\geq j}, \varphi_2)\} + j, \\ &\quad \text{where } j \text{ is the smallest } \geq 0 \text{ s.t. } \langle \sigma, j \rangle \models \varphi_2, \text{ and} \\ &\quad \text{for all } k, i \leq k < j \text{ implies } \langle \sigma, k \rangle \models \varphi_1. \end{aligned}$$

If $\varphi = p \in AP$, then $i(\sigma, \varphi) = 0$ and clearly $\sigma_0 = \sigma_{\leq 0} \models \varphi$, as $p \in \sigma_0$ by hypothesis. The base case for $\varphi = \neg p$ is similar, and the inductive cases for Boolean connectives are immediate.

If $\varphi = \bigcirc \varphi_1$, then $i(\sigma, \varphi) = i(\sigma_{\geq 1}, \varphi_1) + 1$ and by induction hypothesis $\sigma_{\leq i(\sigma_{\geq 1}, \varphi_1)} \models \varphi_1$. Hence, for $i(\sigma, \varphi) = i(\sigma_{\geq 1}, \varphi_1) + 1$, we have $\sigma_{\leq i(\sigma, \varphi)} \models \varphi$.

If $\varphi = \varphi_1 \cup \varphi_2$, then $i(\sigma, \varphi_1 \cup \varphi_2) = \max\{i(\sigma_{\geq k}, \varphi_1)\}_{k < j}, i(\sigma_{\geq j}, \varphi_2)\} + j$ and by induction hypothesis $\sigma_{\leq i(\sigma_{\geq j}, \varphi_2)} \models \varphi_2$ and for all $k < j$, $\sigma_{\leq i(\sigma_{\geq k}, \varphi_1)} \models \varphi_1$. Hence, for $i(\sigma, \varphi) = \max\{i(\sigma_{\geq k}, \varphi_1)\}_{k < j}, i(\sigma_{\geq j}, \varphi_2)\} + j$, we have $\sigma_{\leq i(\sigma, \varphi)} \models \varphi$.

As for the $\Leftarrow$ direction, suppose that $\sigma \not\models \varphi$. Then, by induction on the structure of φ we can show that for every $i \geq 0$, $\sigma_{\leq i} \not\models \varphi$. $\square$

Next we prove that formulae in LTL_f can be translated into *co-safe LTL* in a truth-preserving manner. First of all, we assume operators $\square$ and $\bullet$ as primitive and consider negation on atoms only. We also assume atom *Last* as primitive, only true in the last element in σ . Then, consider translation τ from LTL into *co-safe LTL*, which commutes with all operators, and such that

$$\begin{aligned}\tau(\square \varphi) &= \tau(\varphi) \cup (\text{Last} \wedge \tau(\varphi)) \\ \tau(\bullet \varphi) &= \neg \text{Last} \rightarrow \bigcirc \tau(\varphi)\end{aligned}$$

Proposition 5. Let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite sequence, φ an *LTL* formula,

$$\sigma \models \varphi \text{ iff for every } \sigma' \in (2^{\mathcal{AP}})^\omega, \sigma \cdot \sigma' \models \tau(\varphi) \quad (\text{D.2})$$

where $\cdot$ is string concatenation.

Proof. The proof is by induction on the structure of φ . The case of (negated) atoms, as well as Boolean connectives, is immediate.

For $\varphi = \square \psi$, $\sigma \models \varphi$ iff for every $i \leq |\sigma|$, $\langle \sigma, i \rangle \models \psi$, iff for every $i \leq |\sigma|$, $\langle \sigma \cdot \sigma', i \rangle \models \tau(\psi)$ for every $\sigma' \in (2^{\mathcal{AP}})^\omega$, by induction hypothesis, iff $\sigma \cdot \sigma' \models \tau(\psi) \cup (\text{Last} \wedge \tau(\psi))$ for every $\sigma' \in (2^{\mathcal{AP}})^\omega$, iff $\sigma \cdot \sigma' \models \tau(\varphi)$.

For $\varphi = \bullet\psi$, $\sigma \models \varphi$ iff $|\sigma| \geq 1$ implies $\langle \sigma, 1 \rangle \models \psi$, iff for every $\sigma' \in (2^{\mathcal{AP}})^\omega$, $\sigma \cdot \sigma' \neq Last$ implies $\sigma \cdot \sigma' \models \bigcirc \tau(\psi)$ by induction hypothesis, iff $\sigma \cdot \sigma' \models \neg Last \rightarrow \bigcirc \tau(\psi)$, iff $\sigma \cdot \sigma' \models \tau(\varphi)$. $\square$

By combining Proposition 4 and 5, we obtain the following result.