UNIVERSITY OF SOUTHAMPTON

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

Civil, Maritime and Environmental Engineering Department

Reinforcement learning with limited prior knowledge in long-term environments

by

David M. Bossens

Thesis for the degree of Doctor of Philosophy

May 2020

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
Civil, Maritime and Environmental Engineering Department

Doctor of Philosophy

REINFORCEMENT LEARNING WITH LIMITED PRIOR KNOWLEDGE IN LONG-TERM ENVIRONMENTS

by David M. Bossens

Increasingly, artificial learning systems are expected to overcome complex and open-ended problems in long-term environments, where there is limited knowledge about the task to solve, the learners receive limited observations and sparse feedback, the designer has no control over the environment, and unknown tasks may present at random times to the learner. These features are still challenging for reinforcement learning systems, because the best learning algorithm and the best hyperparameters are not known a priori. Deep reinforcement learning methods are recommended but are limited in the number of patterns they can learn and memorise. To overcome this capacity issue, this thesis investigates long-term adaptivity to improve and analyse reinforcement learning in long-term unknown environments. A first case study in non-episodic mazes with sparse rewards illustrates a novel learning type called active adaptive perception, which actively adapts how to use and modify perception based on a long-term utility function. Such learning systems are here shown to construct emergent long-term strategies to avoid detracting corridors and rooms in non-episodic mazes, where a state-of-the-art deep reinforcement learning system DRQN gets stuck. A consequent case study in lifelong learning, where reinforcement learners must solve different tasks presented in sequence. It is shown that multiple policies each specialised on a subset of the tasks can be used as a source of performance improvement as well as a metric for task capacity, how many tasks a single learner can learn and remember. The case study demonstrates that the DRQN learner has low task capacity compared to an alternative deep reinforcement learning system PPO. The results indicate that this is because PPO's slower learning allows improved long-term adaptation to different tasks. An additional finding is that adaptively learning which policy to use can be beneficial if the policies are sufficiently different from each other. On the same case study, an additional result shows that, when using a long-term utility function to evaluate performance, a correction for the different reward functions is beneficial to avoid forgetting.

Contents

Declaration of Authorship **xi**

Acknowledgements **xiii**

1 Introduction **1**
- 1.1 Learning in long-term environments with limited prior knowledge 1
- 1.2 Artificial General Intelligence 4
- 1.3 Thesis approach 9
- 1.4 Research Aim, Questions & Objectives 10
- 1.5 Contribution and Novelty 11
- 1.6 Thesis outline 11

2 Literature review **13**
- 2.1 Neural networks 13
- 2.2 Traditional Reinforcement learning 17
- 2.3 Reinforcement learning with limited observation 20
- 2.4 Deep reinforcement learning 20
- 2.5 Universalist reinforcement learning methods 27
- 2.6 Other lifelong reinforcement learning approaches 29
- 2.7 Summary 31

3 Long-term adaptivity to alleviate and shift inductive bias **33**
- 3.1 The inductive bias of deep reinforcement learning systems 34
- 3.2 Case studies 35
- 3.3 Long-term adaptivity 37
- 3.4 Summary 47

4 Long-term learning with limited knowledge and sparse rewards **49**
- 4.1 Non-episodic partially observable mazes 50
- 4.2 Active Adaptive Perception implementation 51
- 4.3 Experimental setup 60
- 4.4 Experimental demonstration of active adaptive perception 62
- 4.5 Discussion 75
- 4.6 Summary 77

5 Lifelong learning with multiple policies **79**
- 5.1 Learning many tasks with a limited number of policies 79
- 5.2 Experimental set-up 84

5.3 Results 88
5.4 Discussion 94
5.5 Summary 98

6 Towards active adaptive perception in lifelong learning environments 99
6.1 Lifelong Success Story Algorithm 100
6.2 Experimental setup 111
6.3 Results 113
6.4 Discussion 118
6.5 Summary 120

7 Discussion 121
7.1 Strengths 121
7.2 Limitations 127
7.3 Future work 127

8 Conclusion 131

Appendix A Publications 133

Appendix B Experimental parameters 155
B.1 Learning with limited knowledge and sparse rewards 155
B.2 Lifelong learning with multiple policies 160
B.3 Towards active adaptive perception in lifelong learning environments . . . 160

Appendix C Additional results (Chap. 4) 163
C.1 Effect of duplication 163

Appendix D Additional results (Chap. 5) 165
D.1 Ordinal analysis 165

Appendix E Additional results (Chap. 6) 169
E.1 Lifelong SSC proof 169
E.2 Task-specific development plots 172

List of Figures

2.1 The agent-environment interface for reinforcement learning 18

3.1 Illustration of inductive bias 36
3.2 Block diagram of the generic architecture 40
3.3 Learning with multiple policies 46

4.1 Flow diagram of non-episodic maze with sparse rewards 51
4.2 Implementation of active adaptive perception 52
4.3 Illustration of the network construction operators. 58
4.4 SMP-DRQN's key mechanisms 64
4.5 Heatmaps on a partially observable maze 65
4.6 SMP-DRQN's goal matching 66
4.7 Development of the valid modifications 69
4.8 Development of the valid modifications 70
4.9 Development of the reward speed 72
4.10 Development of the reward speed 72

5.1 Illustration of various tasks based on three defining characteristics of the feature vector. 87
5.2 Illustration of the task sequences 87
5.3 Cumulative reward depending on the number of policies 89
5.4 DRQN performance development on individual tasks, depending on the number of policies 91
5.5 PPO performance development on individual tasks, depending on the number of policies 92
5.6 Policy spread as a function of time, depending on the number of policies . 93
5.7 Effect of intermediate tasks on forgetting, depending on the number of policies 95
5.8 Effect of prior tasks on transfer, depending on the the number of policies 96

6.1 Catastrophic forgetting in SSA 101
6.2 Task-specific reward velocities 102
6.3 Stack structure for Lifelong SSA 104
6.4 The problem with the evaluation-modification-gap 108
6.5 Cumulative reward over time for each evaluation module 113
6.6 Effect of intermediate tasks on forgetting for different evaluation modules 116
6.7 Effect of prior tasks on transfer for different evaluation modules 116
6.8 Development of hyperparameters over the lifetime 117

B.1 Complete set of easy mazes 158
B.2 Complete set of difficult mazes 159

E.1 Performance development of the various evaluation modules on individual tasks 172

List of Tables

2.1 Features of state-of-the-art reinforcement learning methods to deal with challenges related to long-term learning in unknown environments. 26

3.1 Operators on the stack $\mathscr{S}$ and data stored in stack-entries $e \in \mathscr{S}$. 42

4.1 List of instructions for SMP learners 54
4.2 Correctness over time 67
4.3 Variance analysis on the effect of learning condition on lifetime averaged normalised reward speed 73
4.4 Additional performance metrics 74

5.1 Variance analysis on the effect of adaptivity and the number of policies on performance. 90

6.1 List of instructions for SMP learners 112
6.2 Variance analysis on the effect of evaluation module on performance. . . . 114
6.3 Ordinal analysis on the effect of evaluation module on performance. . . . 114
6.4 Effect of task velocities on stack size. 115

B.1 DRQN parameters. 156
B.2 Parameter settings for the base learners 161
B.3 DRQN parameters. 162
B.4 Exploration rate settings' performance 162

C.1 Effect of duplication 164

D.1 Ordinal analysis on the effect of adaptivity and the number of policies on the performance of DRQN. 166
D.2 Ordinal analysis on the effect of adaptivity and the number of policies on the performance of PPO. 167

Declaration of Authorship

I, David M. Bossens , declare that the thesis entitled *Reinforcement learning with limited prior knowledge in long-term environments* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as:
  Bossens, D. M., Townsend, N. C., & Sobey, A. J. (2019). Learning to learn with active adaptive perception. *Neural Networks*, 115, 30-49. *Note: this publication was also featured in Elsevier Press* *https://www.journals.elsevier.com/neural-networks/news/can-ai-ever-learn-without-human-input*.

Signed:...

Date:..

Acknowledgements

This thesis project was only possible with the cooperation of many other important actors. This includes people who organised the PhD grant: Engineering & Physical Sciences Research Council, Lloyd's Register Foundation, my supervisors Adam Sobey and Nicholas Townsend, and others. I thank my supervisors a second time as well for the useful discussions, comments, guidance, and asking the right questions, all of which helped the thesis greatly. I thank the examiners Sarvapali Ramchurn and Ke Chen for their valuable feedback, and I acknowledge the use of the IRIDIS High Performance Computing Facility and associated support services at the University of Southampton.

Chapter 1

Introduction

It has been estimated that by 2030 global GDP could increase by 13.8% merely due to the impact of Artificial Intelligence (AI). AI has become an instrumental part of nearly all human activities and services, with applications in medicine, manufacturing, media, transport, and energy [138]. To solve menial aspects of these domains, software programs can be applied to fullfill specific tasks in specific pre-defined cases known to the designer. However, increasingly, research into AI has been emphasising adaptivity, generality and creativity.

This shift has been motivated by practical demands for intelligent agents performing complex tasks over extended time without human intervention. Entertainment is one key industry, with companies such as Microsoft (Cortana), Apple (Siri), and Amazon (Alexa) developing virtual assistants, and DeepMind providing artificial agents that provide novel and super-human solutions to game playing. Care of the elderly is another consideration, for example, in Japan where the rapidly ageing population must be supported [54]. Another practical consideration is that robotic systems may replace humans in complex activities that are simply too dangerous, including nuclear site disaster response, deep sea missions and space exploration. To illustrate the importance, space economy contribution is currently estimated at 13.7 billion pounds [214], and extensive scientific investigations of other planets are being being planned [38].

1.1 Learning in long-term environments with limited prior knowledge

With increased expectations for general and creative artificial learners, this thesis focuses on scenarios in which these properties are especially important. Such scenarios, best described as "long-term environments with limited prior knowledge", allow low designer effort, novel solutions not foreseen by the designer, and the exploration of the unknown.

1.1.1 Motivating examples

To illustrate the types of scenarios, a few notable examples are here given.

Long-term autonomy in robotics Imagine a future scenario, possibly within a few decades, where robotic devices are fully operational for 200 years or more without requiring recharging or maintenance, and are being used to explore unknown planets. In some such missions help from humans may be sparse at best, since communication is hampered by the delays in signals traveling from one planet to another. Even worse, almost no information will be known about the most distant planets where no human has ever ventured. This implies that, to discover anything meaningful, such robotic devices need to be completely autonomous not only in mechanical operation but also in their learning procedures: the learners have to be sufficiently flexible to peform complex tasks, to set their own goals and to reason across vast temporal scales, all without the help of humans.

Creativity in science, games and artistic fields Productivity in science could be greatly expanded by applying artificial intelligence to the process of science itself. One possibility which is being explored is automated experimentation [184], in which an AI performs the entire cycle of experimentation autonomously. A more ambitious aim is to develop a completely autonomous scientist [234, 63]. One approach would be to design the system pre-built with physical laws. However, there is a possibility that some of science is either wrong or incomplete, and building a system with wrong foundations could prove detrimental. Learning with limited prior knowledge therefore allows more creative solutions, unanticipated by the designers. This, in fact, is analogous to two deep reinforcement learning methods that have made the headlines: the improvement of the Alpha-Go to the Alpha-Zero system [176], which showed that a system trained with expert knowledge actually performed worse than a system which played against itself; the Q*bert-playing reinforcement learning system in [35], which learned how to exploit a bug in the program. Findings such as these show that, even on quite limited time scales, injecting human knowledge into artificial systems can hamper performance and generality. Beyond science and game playing, similar arguments can be made for the arts, where there is a great need for creative solutions.

1.1.2 Characteristics of long-term environments with limited prior knowledge

This thesis will investigate scenarios with long-term environments with limited prior knowledge, which are characterised by the following challenges:

- **open-ended missions**: although humans cannot prescribe what the optimal behaviour is, they can provide feedback on to what extent they like the behaviour. Such open-ended missions may occur when dynamics are unknown or because the aim is vague. For example, exploration missions aim to "discover interesting things" while ethical decisions aim to "improve human well-being".
- **continual lifetime**: the learner is subjected to a single lifetime during which anything that happens may affect what happens later, including environmental dynamics, cognitive processes and decisions. This requires long-range memory of previous events and the ability to explore if they get stuck in a part of the environment in which further learning is difficult. This is opposed to the controlled learning conditions used frequently in AI, in which learning proceeds in episodes of experience with no sequential dependencies and with a time-out to limit the time spent in a single episode.
- **ambiguity of observations**: the observations of the learner may be missing vital information for decision making. This may be due to hidden variables or due to time dependencies within the environment.
- **learning and maintaining skills**: learners must develop transferable skills which can be applied to a wide set of tasks, and avoid losing previously learned skills.
- **efficient use of sparse feedback** when feedback is sparse, the learner needs to make efficient use of the limited information received about the goodness of its behaviour.

Beyond the mentioned bullet points, there are additional challenges not considered in the thesis which would characterise realistic long-term robot missions. This includes: (a) fault-tolerance and robustness in case robots are damaged or experience other types of failures; and (b) feedback conveying data about the environment or the robot rather than the goodness of the robot's current behaviour.

To deal with learning with limited knowledge from experience, a common framework that is suggested is **reinforcement learning**. The complexities that follow from reinforcement learning in unknown long-term environments come in at least two categories:

- A single task is difficult to learn, either because the learner cannot focus its exploration on the most useful patterns, because its representation cannot capture the patterns, or because the algorithm does not apply the learned patterns correctly.
- A multitude of tasks is difficult to learn because knowledge about one task may affect adversely the performance on another task.

These two categories form the core of the thesis' investigations. In light of these difficulties, the designer cannot pre-determine the best strategy, the best hyperparameters or even the best learning algorithm, and new tasks have to be learned by experience. This is a challenge even for state-of-the-art reinforcement learning systems.

1.2 Artificial General Intelligence

As a solution to such long-term, open-ended and complex problems, the designer cannot hand-craft a solution for each problem and then inform the agent. The artificial agents must therefore have *general intelligence*.

1.2.1 Definitions of general intelligence

A key concept in psychological literature is general intelligence or g for short [185], the potential to perform well across seemingly uncorrelated tasks, by virtue of a general cross-domain reasoning and memory capacity. Over the years, psychology has provided metrics such as IQ tests, not without its critics [190], and biological metrics based on brain sizes [29]. However, when considering artificial agents, a more objective and also less anthropocentric definition of machine intelligence was given by [105], where a mathematical value is calculated based on achieving high cumulative rewards over a vast amount of time over a wide set of environments – with the simplest environments having a greater weight due to the principle of algorithmic probability which is similar to Occam's razor. Goertzel [62] extends this definition to incorporates a probability distribution over resources such as memory, time and energy, and then penalises those agents that consume too much of them. Similarly, Wang [223] considers the Assumption of Incomplete Knowledge and Resources (AIKR), which states that resources such as time, memory and knowledge are limited, as the key assumption behind general intelligence. When the AIKR is not met, it is often more productive to use Narrow AI methods specialised to the given problem or mathematically optimal solvers instead of AGI; for example, for any task with a well-defined optimal strategy, a sufficient amount of resources, and all relevant information, the mathematically optimal solver can be applied without any problem, and, by definition, it will yield the best results.

1.2.2 Background on AGI

From the above it is clear that machine intelligence metrics typically emphasise the ability to obtain high rewards in a wide set of environments and with limited knowledge and resources. In *reinforcement learning*, an agent interacts with the environment to obtain high cumulative reward. This makes reinforcement learning a popular approach for open-ended environments and AGI.

Historically, reinforcement learning can be traced back to the early years of psychology, with Pavlov's classical conditioning [133]: when a particular stimulus was consistently paired with a so-called Unconditioned Stimulus, a stimulus that evokes reflexive responses, the mere sight of this stimulus, the Conditioned Stimulus, will evoke this reflexive response. For example, a dog will salivate when it smells meat, but by repeatedly coupling the sound of a bell together with meat, this sound will start evoking salivation as well. A related principle called operant conditioning was found by Thorndike [208] and Skinner [182]: by incurring a reward (positive reinforcement) or punishment (negative reinforcement) it is possible to stimulate particular responses. For example, a mouse may be given food whenever it presses a lever, thus causing the mouse to press the lever whenever it needs food.

Other important foundations for AGI were soon to follow. Kurt Goedel, in a study of self-referential statements, had shown that there are essential limits to the statements that may be proven in formal systems [61]. Alan Turing had created a formal structure, called the Universal Turing Machine, a general purpose machine which could simulate all other machines [212]. Inspired by the workings of the human brain, a first mathematical model for neural networks was formulated by McCulloch and Pitts [120]. In 1950, Turing proposed a test for assessing whether or not a machine is as intelligent as a human, the idea of which was to test whether or not a machine could credibly imitate a human being [213]. In 1956, the Dartmouth Summer Research Project on Artificial Intelligence [119] first coined the term "Artificial Intelligence" (AI) with the aim of creating self-improving creative machines with thinking capacities similar to humans. In 1957, Rosenblatt [141] invented the perceptron algorithm, which made it possible to learn binary classification by adaptively changing the weights of a linear function, and was optimistic this system would achieve human-like intelligence. Around the same time, reinforcement learning was formalised in the dynamic programming of Bellman [19] where a probabilistic model of state-transitions calculates the optimal action given the current state, based on a long-term value function which accumulates the rewards over a number of time steps. The term AI was used in the 50s mainly for the pursuit of human-like general intelligence, and in the sixties, researchers were optimistic that human-level AI would be achieved within several decades. However, the book 'Perceptrons' by Minsky and Papert in 1969 showed the limits of the single-layered perceptron [122], and the neural network approach was abandoned for a while.

Between 1970 and 1990, it had become apparent that "AI" had not lived up to the hype; consequently, two periods, commonly referred to as "AI winters", occurred in which AI funding was reduced. This also lead to new fields being investigated. The 70s and 80s had lead to the emergence of the first cognitive architectures, such as ACT [3, 4] and Soar [101], which mimic the core functionalities of human cognition by the use of symbolic reasoning. Concept learning, the induction of general rules from limited number of examples, was especially undergoing rapid development. Many of these studies focused on inductive bias, assumptions in the learning representation or the learning algorithm that affect the generalisation. This highlighted one of the key properties of induction: a strong bias enables rapid learning of a limited number of patterns while a complete lack of bias would amount to simply restating the training examples or making all generalisations equally likely, not learning at all [124].

In 1986, the work in [143] popularised the backpropagation algorithm for neural networks, by demonstrating how hidden layers can form abstract representations. Another notable contribution for formalising reinforcement learning was Watkins' Q-learning [225], a method which allowed to learn in environments for which the learner has no model. From 90s onward, neural networks gained more popularity, and with Moore's Law enabling doubling computing power every 12-24 months, large networks with many layers, deep neural networks, eventually became feasible, and evidence of its increasing scalability of neural networks and reinforcement learning is already seen in 1995 in the work of Tesauro, with a human-level performance at the Backgammon game [206]. Together with algorithmic advances, deeper neural networks allowed impressive performance on tasks difficult for other AI techniques, such as image recognition and natural language processing. When reinforcement learning was later combined with deep neural networks, the field of *deep reinforcement learning* has made news headlines several times due to exciting robotics applications, such as learning from limited feedback how to manipulate objects [1], and super-human performance in games such as Shogi, Chess, and Go [83].

In present times, many scientists use the term AI to describe endeavours that do not explicitly aspire to emulate the generality of a human: rather than attempting to mimick human-like general intelligence, scientists researching AI investigate specific issues such as the optimisation of the cost of an engineering design [115] and classification tasks [216]. However, a growing number of scientists have regained optimism in the achievement of human-level intelligence, and this is reflected in the birth of a new field called Artificial General Intelligence (AGI). The opinion of AGI scientists is paraphrased best by the Core AGI hypothesis [63], which states that

> *"the creation and study of synthetic intelligences with sufficiently broad (e.g. human-level) scope and strong generalization capability, is at bottom qualitatively different from the creation and study of synthetic intelligences with significantly narrower scope and weaker generalization capability."*

1.2.3 AGI approaches

Having given a background to AGI, this section describes the various modern approaches to AGI, according to the categorisation of [63].

Symbolic approach The symbolic approach relies on the notion of a "symbol", a compact and discrete representation of a larger concept whose meaning derives from the programmer. In this approach, a language expresses declarative knowledge about the real world and various procedures to manipulate and grow knowledge bases. The learners can then reason about these fact based on logic, inference, and if-then-rules, also called production rules. Unlike traditional Logical AI, these learners allow the revision of beliefs using non-monotonic logic. Various cognitive architectures have been proposed in this category [103, 93, 173, 107, 34] and generally these are not using reinforcement learning. However, Soar, one of the earliest but still active cognitive architectures has been extended to include reinforcement learning to help procedural learning [100] when no production rules are applicable. The symbolic approach, with its logical reasoning, is good for: well-defined problems and high-level decision-making; using a database of facts, it is not prone to catastrophic forgetting, in the sense that many of the facts remain constant during the lifetime and that, in case revision is required, new facts may be created and different facts may be merged together to a more general fact; the language and basic symbols make it transparent to the designer. However, its key limitation is the symbol grounding problem [194, 189, 204, 68]: their symbols are not "grounded", meaning they are not derived from sensori-motor interactions with the world, and this implies not only that such systems rely heavily on designer knowledge but also that they may not always be able to identify what the symbol is referring to and to form a fully connected network of meanings. Further, its if-then rules do not sufficiently take into account uncertainty, and its low-level pattern recognition and real-number processing are limited. The approach is also prone to memory overload due to requiring a huge database of facts.

Sub-symbolic approach Inspired by biological organisms, the philosophy of the sub-symbolic or emergentist approach is that high-level concepts will emerge from low-level elements. The approach is also connectionist because meaning arises only from the connection of the elements, whereas a single element does not have any inherent meaning. They are characterised by numerical computations and parametric adjustments rather than logic inference. Some such approaches attempt to reverse engineer the brain [71, 90], others use a developmental robotics [7, 239, 148] approach. Many of these approaches are heavily inspired by reinforcement learning with intrinsic motivation [195, 153, 181], lifelong reinforcement learning [20, 211, 180, 139], and developmental psychology [137]. Most of the deep learning and deep reinforcement learning methods [6, 104, 108, 163] could also be classified in this category due to their reliance on the connectionist representation and their increasing emphasis on generality. End-to-end RL has emphasised that a variety of functions can emerge even from utilising a single neural network for deep reinforcement learning [174]. Strengths of the sub-symbolic approach include: the ability to work with real numbers; to recognise general patterns in high-dimensional spaces and across time, making them suitable for perceptual processing. Limitations include: opaqueness, meaning it is often difficult to understand how they work and why they do or do not perform well on a particular problem; sensitivity to tuning parameters and initialisation; they are often sensitive to catastrophic forgetting.

Hybrid approach The hybrid approach combines insights of the symbolic and the sub-symbolic approach. This has been implemented in various cognitive architectures [57, 56, 64, 197, 31]. A large number of the works in this approach can be found under the name neural-symbolic learning or neural-symbolic reasoning [22]. The amount of research in reinforcement learning in this line of research is scarce, although recently methods for deep symbolic reinforcement learning [60, 41] have been proposed. The strengths and limitations of this approach are in part explained in the symbolic and sub-symbolic approach; however, an additional difficulty in the hybrid approach is how to integrate efficiently the sub-symbolic and the symbolic level without a loss of generality or information.

Universalist approach The universalist approach, in contrast to the sub-symbolic approach, is not at all biologically inspired but rather is based on the idea that a single meta-algorithm, if applied recursively, will be able to find a good program. Such meta-algorithms construct better and better solution algorithms to problems in their environment. In reinforcement learning problems, this results in an algorithmic inductive bias that facilitates more rapid learning of the correct behavioural patterns. Two typical algorithms are mentioned in Goertzel's classification, both of which are reinforcement learners: the AIXI [81] which maximises future reward with a simplicity bias which weights world models' predictions based on their algorithmic complexity, and the Goedel Machine (GM) [160, 162]. The GM proposes modifications to itself: its axioms, its proof search, or its policy. Only if the GM can prove mathematically that these modifications result in an improved expected utility, the modifications are performed. The GM is a *self-modifying policy*, a reinforcement learning policy which includes in its action set instructions which modify the policy. Whilst providing a method for self-improvement, such learners are bounded by the limits to the provability within complex systems as proven by Goedel's incompleteness theorems. Related methods are those that allow flexible search in program space but are not based on reinforcement learning [223, 131, 159]. The main strength of the universalist approach is the flexibility to construct arbitary learning programs from experience, representing a true meta-learning, the ability to learn how to learn. The limitations are the lack of biological plausibility, the scalability issues due to the search space being increased when the learning algorithm is included in the learning process, and the fact that the approaches remain mostly theoretical.

1.3 Thesis approach

Based on the definitions of general intelligence in Section 1.2.1, a likely ingredient characterising the study of general synthetic intelligence, as referred to by the Core AGI hypothesis, is reinforcement learning in long-term environments with limited prior knowledge. This characterises the approach taken in this thesis.

One argument that forms the foundation of the thesis is when there is a lack of prior knowledge, there are certain *core assumptions* that can be made to help reinforcement learners obtain a favourable bias towards the tasks they face. Some may argue that, if the task or tasks are not known by the designer, there is no a priori distinction between algorithms and artificial general intelligence is impossible. Representative of this view, the no free lunch theorems [233] state that the performance of all optimisation algorithms is the same when averaged over the set of all problems, assuming that problems are sampled *iid* from a uniform distribution. However, as commented in [159], the existence of regularities across successive tasks could potentially be exploited. This could be done by transferring knowledge gained in one task to another task, by exploiting a structured curriculum, or even by learning how to learn. Moreover, even in cases with limited prior knowledge, some features of the system and the environment are known at design time. For example, regardless of their task, robotic systems with pixel-maps as sensory inputs will need to perform visual recognition and, due to their spatial inductive bias [123], the assumption of local correlations in spatial structures, convolutional neural networks achieve elevated performance on such tasks. Similarly, when a robot's sensors are limited, it is beneficial to apply algorithms which take into account historic information rather than only the current sensory observation.

Another position taken by the thesis is to emphasise the role of adaptivity to improve the generality of reinforcement learners, but to recognise the possible trade-offs between adaptivity and efficiency. Defining adaptivity in terms of the type and the number of parameters that can be modified by the learning algorithm and efficiency in terms of the number of samples required to solve a problem, the following trade-off is typically observed: on the one hand, adaptive systems may be slower to find the optimal solution since they require expensive experience with the environment; on the other hand, adaptive learners can solve a wider variety of problems with a larger ceiling on performance. When solving a single task, the unadaptive learner may have a limited representation that cannot capture a valid solution; its algorithm may make certain assumptions that are only true for the limited cases anticipated by the designer; and, there may be solutions not envisioned by the designer. Further, when solving a large number of unforeseen tasks, the unadaptive learner cannot be pre-programmed to solve all tasks. The adaptive learner, by contrast, may improve as time proceeds, as transfer of knowledge may exploit similarities among the tasks, and as meta-learning can be used to exploit rewarding patterns in the learning process itself.

1.4 Research Aim, Questions & Objectives

The aim of the research is to investigate long-term adaptivity to enable reinforcement learning with limited knowledge in long-term environments. To address this issue, the thesis targets the following questions:

- How can adaptivity be exploited to provide improved learning in unknown environments?
- What is the trade-off between adaptivity and efficiency?
- Which kind of loss or utility functions are useful to guide reinforcement learning in long-term environments?
- How prone are reinforcement learning policies to forgetting in lifelong learning environments?
- How well do reinforcement learning policies transfer across tasks in lifelong learning environments?

Consequently, the thesis pursues the following objectives:

1. **Literature review**: To review the literature on reinforcement learning with limited prior knowledge, sparse rewards and a variety of unforeseen tasks, and identify gaps in the state-of-the-art.
2. **Learning a long-term task with limited knowledge and sparse feedback**: this objective has three measurable components:
   A to demonstrate the capability of a novel adaptive reinforcement learning system to learn how to learn with limited assumptions, based on a long-term utility function
   B to obtain favourable performance in a long-term task with limited knowledge and sparse rewards, when compared to state-of-the-art deep reinforcement learners.
   C to provide a detailed analysis of the key factors to the performance of this system.
3. **Learning multiple tasks in sequence**:
   A to propose adaptive algorithms to improve lifelong learning, including avoiding catastrophic forgetting and improving selective transfer.
   B to provide a theoretical and empirical analysis on utility functions to improve lifelong reinforcement learning.
   C to analyse the scalability of reinforcement learners to a large number of tasks and a longer lifetime.

1.5 Contribution and Novelty

Most existing work on reinforcement learning assumes relatively well-behaved environments: the observation of the environment perfectly describes the state of the environment, the reinforcement learning agent receives frequent feedback and can be reset whenever a trial does not work out, and there is usually only a single task, which ignores the changing requirements of the environment. By contrast this thesis focuses on environments in which these assumptions do not hold true. Two challenging case studies are considered, one in which there is a prolonged task in which the learner may get stuck without any feedback at all, another in which there are 18 unique tasks sampled randomly and presented in sequence.

To solve such ill-behaved environments, the thesis investigates the role of *long-term adaptivity* in overcoming the challenges in such environments. The thesis defines long-term adaptivity as augmenting traditional reinforcement learning methods with a selective use of representation learned by long-term experience with the environment. To allow adaptation of the use and learning of perception based on a long-term utility function, and deal with unknown environments some of which may not be solvable by deep reinforcement learners, a novel meta-learning principle, called active adaptive perception, is proposed and implemented using a unique combination of universalist and sub-symbolic AGI. The method is highly adaptive by including learning operations as part of the parameters, and represents a trade-off between adaptivity and expressive efficiency; an initial phase of learning to learn is required to find a suitable learning strategy, but eventually the learning strategy allows exceeding the performance on ill-behaved environments due to finding an alternative learning algorithm more suitable to the environment than traditional reinforcement learning. To improve performance in lifelong learning scenarios such as the above-mentioned 18-task scenario, a multiple policy reinforcement learning approach is proposed for lifelong learning scenarios, where a policy is selected based on the long-term cumulative reward on the task. The approach is proposed both as a tool to study the goodness of a lifelong reinforcement learner as well as a possible alternative approach for lifelong reinforcement learning. The method improves selective transfer and reduces catastrophic forgetting by defining policies across a subset of tasks and allows to study different base-learners with different utility functions. Finally, to allow reinforcement learners to learn with a long-term objective with reduced catastrophic forgetting in lifelong learning environments, a novel long-term utility function is proposed which corrects for the different reward functions of different tasks.

1.6 Thesis outline

The thesis first investigates the existing literature on reinforcement learning in long-term environments in Chapter 2. The general thesis methodology is provided in Chapter 3.

Chapters 4-6 are the contribution chapters: (i) Chapter 4 investigates the use of active adaptive perception for a long-term single task scenario where the learner may get stuck without any feedback; (ii) Chapter 5 investigates learning with multiple policies in an 18-task scenario; (iii) Chapter 6 investigates a novel objective function to allow active adaptive perception in lifelong learning scenarios. These contributions are then discussed in Chapter 7, and the thesis concludes in Chapter 8.

Chapter 2

Literature review

This chapter investigates closely the literature on reinforcement learning in long-term unknown environments. Key difficulties in such environments may be: hidden variables due to the limited observation; changes in the tasks provided to the learner; sparse feedback; limited control over the learning conditions, for example for when to finish a task. Because of the need to overcome a large amount of unknowns, this challenge raises fundamental questions about the adaptivity, capacity and bias of learning systems.

2.1 Neural networks

Perception [149] is defined as the organization, identification, and interpretation of sensory information in order to represent and understand the presented information, or the environment. The perception of a learner must be adaptive because the demands of the environment can change. Neural networks, which are easily integrated with with reinforcement learning algorithms, are probably the best known example of such *adaptive perception*, as they are able to learn from experience various functional regularities.

2.1.1 Types of neural networks

Feedforward neural networks takes an input x, processes it through its hidden layers of neurons, and at the output layer it produces an output y. They are especially used in static classification tasks, like image recognition. It is well known that deep neural networks can represent complex hierarchies in a manner similar to that of the visual cortex [163]; convolutional neural networks [65, 66, 85, 91, 97, 98, 99, 113, 186, 202, 237] have been especially useful in such tasks.

Recurrent neural networks add a temporal dimension to the network, allowing them to learn dynamic systems. At a step t, a recurrent neural network takes an input x_t, propagates x_t through the hidden layers with feedforward connections but also propagates neuronal activations from time $t-1$ through the hidden layers using recurrent connections, after which an output y is produced. Popular examples include Elman networks [45], fully recurrent neural networks [144], Gated Recurrent Units [33], and Long Short-Term Memory networks (LSTMs) [77]. Amongst these LSTMs have by far yieldest the most impressive performance results. For example, they have been applied at speech recognition and similar problems with continuous streams of information [48, 227, 48, 67], where due to their forget gates they can learn when to forget certain information. The LSTMs ameliorate the vanishing gradient problem by including a self-recurrent unit which continually repeats past information [77]. Recurrent neural networks can be used in addition to convolutional neural networks, see for example [69]. Additional flexibility in representation building can be achieved by further adding the ability to determine the neural network's architecture, as is done in constructive neural networks [109, 46, 58].

2.1.2 Theoretical capacity

A key subject of study for theoretical investigations has been the capacity, or expressiveness, of neural networks, which loosely phrased means the number and type of functions it can represent. Over the years, there have been a variety of results in this regard. In the context of binary classification, one of the tools to analyse capacity is the Vapnik Chervonenkis (VC) dimension [220], which considers the largest number of points for which it would be able to classify each possible combination of labels accurately. Specific for feedforward neural networks, a polynomial formula was given for the capacity of feedforward neural networks with linear outputs in [13], which was based on the sizes of the different layers. With metrics such as these, the general conclusion is that capacity is increased by including the number of neurons and parameters and by including non-linearity in the function. Moreover, the capacity and sample complexity are proportional, such that more expressive learners also need more training data [183]. This trade-off has also been phrased as expressive efficiency [36]: as the search space more closely fits the true function, the learner will be less expressive but much faster to obtain the solution. This is also embodied in Probably Approximately Correct (PAC) learnability [217], whether or not an algorithm can learn, within a number of N samples, a solution with an error of at most ϵ with a probability of at least $1-\delta$, for some $\delta > 0$ and some $\epsilon > 0$ – that is, VC dimension and the upper bound on sample size N required for PAC-learnability are inversely related.

Neural networks have strong theoretical results about their capacity, backing up their succcessful practical application. One of the earliest theoretical results is the proof that feedforward neural networks are universal function approximators [79], when they have at least one hidden layer with sufficient amount of sigmoid-type units. This result has later been extended to yield even more impressive results for recurrent neural networks, which were shown to be able to approximate any open dynamical system with arbitrary efficiency [150].

2.1.3 Training methods and their practical difficulties

Despite the impressive theoretical results, neural networks are by no means universal in practice. One important issue is that the data acquired must fairly represent the function to be approximated and that they require much training time due to their many free parameters. The training procedure itself also defines an inductive bias, preferring some generalisations over others. These aspects are discussed further below for the two most popular tools for training neural networks, gradient descent and evolution, and for a novel emerging framework based on meta-learning.

Gradient descent Gradient descent methods calculate an objective function on the outputs and then update the interconnections between the neurons in the direction of the gradient, effectively performing a local search. For this reason, such algorithms tend to get stuck in local optima if the of the objective function is deceptive. The backpropagation algorithm [143] has been traditionally used for feedforward neural networks. Recurrent neural networks such as those mentioned above are often trained using backpropagation-through-time [127], and due to multiplication and addition, this can lead to the gradient either vanishing, resulting in no updating, or exploding, resulting in too strong updates [21]. Although LSTMs do not tend to suffer from the vanishing gradient compared to traditional recurrent neural networks [77], comparable issues may occur due to the exploding gradient [199]. Using backpropagation-through-time, learning across large history windows can be time consuming, and the window also limits the time period across which can be memorised, but has the advantage of being able to be learned offline, meaning the data provided to the learning algorithm do not necessarily have to be the data that is currently being gathered. Compared to back-propagation, real-time recurrent learning methods [230] can accumulate the gradient over time incrementally without a limiting time window, and a forward propagation step requires only a single input to the network rather than a history window. This allows them to bridge longer time gaps without excessive computational expense, but at the expensive that all data must be provided online and therefore this type of algorithm is more sensitive to overfitting and getting stuck in a local optimum. To this day, gradient descent methods are the most popular training method for neural networks because of how they allow fast training of large networks. Inherently, there is no real adaptivity in the gradient descent type of methods. A type of pre-programmed adaptivity is included in adaptive gradient methods [94, 238], which adjust the size of the update to compensate for the sizes of the previous gradients for each weight. The main empirical results provide evidence that the effect is to speed up learning, but sometimes at the cost of reduced generality [232].

Evolutionary algorithms Evolutionary algorithms come in many flavours, but a key principle is that they evolve a population of individuals, each of which represents a solutions to the given optimisation problem. This idea has also been applied to training the weights of a neural network [82, 52, 196]. Compared to gradient descent type methods, a downside is that they are slower to train, not only because of the many advances in efficient differentiation methods, but also because evolutionary algorithms are a form of random search. However, one major advantage over the gradient descent type of methods is that by maintaining a population it is less likely to get stuck in a local optimum and to learn in more deceptive objective functions. Unlike gradient descent methods, evolutionary algorithms allow to evolve neural network topologies. This can be done by modifying a direct encoding of a neural network, as is done in NeuroEvolution of Augmenting Topologies (NEAT) [188], or indirectly, based on a generative encoding of a neural network (e.g., [187]).

Unfortunately, it must be assumed that evolutionary algorithms are done in simulation rather than in online fashion, because each individual must be evaluated in the same type of trials. Since the main aim of this study is to investigate an agent's learning development throughout a long-term autonomous lifetime, rather than a controlled simulation of perfectly repeatable trials, evolution will not be further considered in this literature study.

Meta-learning A less explored alternative for training neural networks is meta-learning, where the training process itself is subject to learning. By adapting the algorithm, this gives them a higher capacity in practice, even when representationally equivalent. In contrast to the above-mentioned training methods, most of these methods have not yet seen application to reinforcement learning. The most influential example in this regard is the class of hyper-networks [5, 78, 40, 116], in which a top-level super-ordinate network searches, by gradient descent, for ways to update the weights of a sub-ordinate network. Such systems are not yet true meta-learning systems, as they are limited to a single meta-level; there is one system which optimises one other sub-ordinate system. Systems with unlimited meta-levels would allow optimisation of the super-ordinate system, the super-super-ordinate system, etc. Such systems are *self-referential*, i.e., they take their own data and algorithms as input to their learning. This type of system has been explored in the self-referential weight matrix [154], which has the ability to take all its adaptive parameters, including those parts of the weight matrix which modify and analyse the weight matrix. This method is far from ready for practical application; a remaining challenge for meta-learning is to realise practical self-referential learning systems. Finally, there are other works that use the term meta-learning to denote the use of a meta-level objective without modifying the bottom-level training algorithm. Such systems have been explored in the reinforcement learning context; for example, for multi-task reinforcement learning, Model-Agnostic Meta-Learning [51] uses a shared parameter vector as an initialisation for a new batch of tasks. From a computational perspective, such methods can be quite efficient, however, they do not share the same flexibility to modify the learning algorithm as hyper-networks or self-referential networks.

2.2 Traditional Reinforcement learning

This section presents the traditional reinforcement learning approaches which revolve around the Markov Decision Process framework. This section describes these formalisms and some well-explored issues within the framework. This mainly helps to lay the foundations for the later literature which builds further on these issues and formalisms.

2.2.1 Reinforcement learning formalisms

Agent-environment interface Most reinforcement learning algorithms are derived from the Markov Decision Process [18] formalism. In Markov Decision Processes, there is a set of environment states $\mathcal{S}$, a set of actions $\mathcal{A}$, a model for state transitions $P(s_{t+1}|s_t, a_t)$, with $s_t \in \mathcal{S}$ and $A_t \in \mathcal{A}$, and a reward function $r(s_{t+1}|s_t, A_t)$. In reinforcement learning the same formalism is used but the state transition model is assumed to be unknown, and is derived only implicitly from interacting with the environment. These interactions take place based on a repeated cycle: at a given time step t, an agent (i) receives an input $s_t \in \mathcal{S}$; (ii) performs an action $A_t \in \mathcal{A}$, where $\mathcal{A}$ is the action set; (iii) receives a real-valued reward $r_t \in \mathrm{R}$. The agent's action selection is based on a **policy** $\mathcal{P} : \mathcal{S} \to \mathcal{A}$, and the agent's aim is to find a policy which maximises a particular utility function. An example of a utility function is the discounted cumulative reward:

$$U(t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}\,, \tag{2.1}$$

where $\gamma \in [0, 1)$ is the discounted cumulative reward. In most reinforcement learning set-ups the environment is considered in discrete **episodes**; this means that the environment is started in some initial state, then runs for a while performing the above-mentioned cyclic process, and then halts as a terminal state is reached, to start again in an initial state, etc.

Reinforcement learning approaches and their fundaments Approaches to reinforcement learning are generally classified into three approaches. **Value-based methods** capture for each environment state how good the action is by mapping a state-action pair to a particular utility function, usually reflecting the discounted cumulative future reward $U(t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ obtained when following a particular state-action trajectory. This works well when the action space is discrete. Value-based methods originated in the

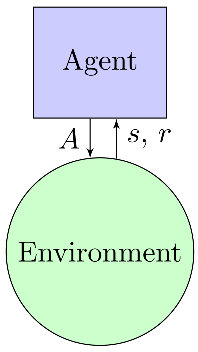


Figure 2.1: *The agent-environment interface for reinforcement learning. The agent continually performs a cycle in which it registers information from the environment via its sensory inputs, namely a state s and a real-valued reward r, and then outputs an action a.*

nineties, and well-known examples are Q-learning [225, 226], State Action Reward State Action (SARSA) [144], and Advantage learning [10]. The value-function of Q-learning [225, 226] is given by :

$$Q(s_t, A_t) = (1-\alpha) * Q(s_t, A_t) + \alpha(r_t + \gamma \arg\max_{A_{t+1} \in \mathcal{A}} Q(s_{t+1}, A_{t+1}))\,, \tag{2.2}$$

where s_t is the current state, A_t is the current action, r_t is the current reward, s_{t+1} is the next state, A_{t+1} is the next action, and $\gamma \in [0,1]$ is the discount factor for the discounted future reward, and $\alpha \in [0,1]$ is the learning rate. Traditionally, the Q-values are stored in a matrix in which the columns represent the states and the rows the actions. The SARSA [144] updating algorithm is similar to Q-learning, but instead of updating the Q-value towards the optimal policy, the updating is done towards the action A_{t+1} actually taken at time $t+1$:

$$Q(s_t, A_t) = (1-\alpha) * Q(s_t, A_t) + \alpha(r_t + \gamma Q(s_{t+1}, A_{t+1}))\,. \tag{2.3}$$

Related to the Q-function is the advantage function $\mathbb{A}$ [10, 9]. Defining the best possible Q-value for a given state s_t as $Q^*(s_t) = \max_{A \in \mathcal{A}}(Q(s_t, A))$, it can be seen as the advantage of a particular action A_t for a given state s_t:

$$\mathbb{A}(s_t, A_t) = Q^*(s_t) + (Q(s_t, A_t) - Q^*(s_t)) * k/dt\,, \tag{2.4}$$

where dt is the user-defined size of a time step, controlling the continuity for continuous-time applications, and k is a user-defined constant. Advantage learning is sometimes used as an alternative to Q-learning because it allows custom time steps and because it can distinguish better the action values when there are small differences between Q-values. **Policy-based methods** directly update the policy's parameters θ based on a score function, a measure of the quality of the policy's parameters. Unlike most value-based methods they are suitable for continuous or stochastic actions, and they do not require an explicit estimate of the value of each action in each state. One popular class of policy-based learners is the policy-gradient methods. REINFORCE [229] is a traditional policy gradient method which updates its policy parameters according to $\theta = \theta + \nabla_\theta \log(\mathcal{P}(A_t|s_t;\theta))(U_{obs}(t) - b(s))$ with $\mathcal{P}$ being the policy, $U_{obs}(t)$ the observed discounted cumulative reward in the entire episode from time t onwards, and $b(s)$ a reward baseline for a given state; this effectively follows the gradient to obtain parameters that maximise the discounted cumulative reward. This method also serves as an example of Monte Carlo updating, a type of evaluation process which simply averages the discounted cumulative reward across the episode, and which can also be used in value-based methods. **Actor-critic methods** are now quite simple to explain: they rely on the interaction between an actor, a policy-based method which learns a distribution over actions according to a policy $\mathcal{P}(A_t|s_t;\theta)$, and a critic, a value-based method which critically evaluates the actions taken by the actor according to some value-function $V(s_t)$. This allows them to use a value-based representation whilst learning continuous and stochastic action policies. Although these are in a sense a combination of the previous two types, their roots trace back to much earlier works [14].

Exploration vs Exploitation A common example to illustrate the importance of exploration is the k-armed bandit setting [8], a stateless setting which is analogous to selecting one machine from a multitude of gambling machines each with their own pay-off distribution. In such setting, should the agent continue selecting the action with the best observed pay-off, or explore other actions the value of which, due to the stochasticity, may have been underestimated ? To solve this dilemma, a few basic strategies are common in value-based methods: (a) optimistic initialisation, with initial value functions set to high values such that the unexplored observation-action pairs will be more frequently visited; (b) epsilon-greedy exploration, in which the explorer selects the policy's best action with a probability $1 - \epsilon$ and a random action with probability ϵ, (c) Boltzmann exploration, which selects an action probabilistically depending on the action-values for a given observation, $p(A_t|s_t) = \frac{\exp(Q(s_t,A_t))}{\sum_{A_t \in \mathcal{A}} \exp(Q(s_t,A_t))}$; (d) an exploration bonus for states that are less frequently visited. Policy-based methods can avoid additional exploration mechanisms like the above, because they can directly optimise their policy to be more deterministic or more stochastic, and the advantage there is that this allows an exploration component sensitive to the current state.

On-policy vs off-policy Compare the Q-learning update, given in Equation 2.2, to that of SARSA, given in Equation 2.3; the former updates its parameters based on the best possible action whilst the latter updates based on the action that was actually taken. Q-learning therefore represents *off-policy learning* whilst SARSA is illustrative of *on-policy learning*. The off-policy learning methods can in principle learn from data collected at any time in the past and their behaviour policy can be different from their greedy policy, whilst on-policy cannot; on-policy methods allow learning stochastic policies whilst off-policy methods must rely on some exploration parameters.

2.3 Reinforcement learning with limited observation

The bulk of the reinforcement learning literature assumes the *Markov property*: the transition probabilities in the environment satisfy

$$P(s_{t+1}|s_t, A_t) = P(s_{t+1}|s_t, A_t, s_{t-1}, A_{t-1}, \dots, s_0, A_0)\,. \tag{2.5}$$

Realistically, any agent can only partially observe the world with its limited sensory capacities.

To solve this problem, the traditional Markov Decision Process framework has been extended to *Partially Observable Markov Decision Processes*. In this case, reinforcement learning agents do not get the environment state as input but rather they receive a limited observation o_t from an observation space $\mathcal{O}$. The observation space is ambiguous, meaning the same observation can map to many different environment states.

Some of these approaches try to reformulate the problem to a Markov Decision Problem. This approach makes use of a so-called belief state $p(s_t|h_t)$, where h_t is the action-observation history and s_t is the state [87, 179, 172]. These solvers still assume knowledge of the underlying variables that determine the state uniquely, although the learner never knows the variables' exact values, and they assume further that the underlying state-dynamics are Markovian. A more general scenario, perhaps best called non-Markovian problems or partially observable environments, is a scenario in which the learner does not know which variables are underlying the environment's state and in which in principle the environment could itself be non-Markovian.

An alternative category which does not require this assumption is tree-based reinforcement learning; this consists of methods, such as U-Tree [118] and Active-LZ [47], which roll out a tree over observation-action histories. Such methods cannot learn beyond a fixed depth of the observation-action history. Also, for complex or continuous observation and action spaces the tree would be large or infinite, making the approach unscalable.

2.4 Deep reinforcement learning

To really obtain scalable methods for dealing with limited observation and other challenges related to unknown environments, deep reinforcement learning integrates the pattern recognition abilities of deep learning with the motivational structure provided by reinforcement learning. Here, state-of-the-art deep reinforcement learning methods are discussed in terms of how they learn in unknown long-term environments. The structure of the section will follow the categorisation used in Section 2.2.1.

2.4.1 Value-based learners

Since the early days, in which a look-up table records for each state and each action an associated Q-value, scalability of value-based methods has increased by using a deep neural network to approximate the Q-function, taking as inputs the observation and action and outputting the Q-value. This has lead to Deep Q-Networks (DQN) [126] which achieved human-level performance across several Atari games. DQN's performance results are attributed to solving two issues in approximating the Q-function: (a) there is a correlation between the Q-values and the target such that, if the Q-value is adjusted, so does the target, causing instability; (b) observations are correlated over time and changes to the policy will further affect this data distribution, making online learning problematic. To resolve (a), a distinction is made between the actual Q-function, which is being continuously trained, and the "target network", which at time t provides the target $r_t + \gamma Q(s_t, \arg\max_{A_{t+1} \in \mathcal{A}} Q(s_{t+1}, A_{t+1}))$ in Equation 2.2, is updated only periodically after thousands of time steps. To resolve (b), Q-learning is supplemented with experience replay [111], a technique in which learning experiences (s_t, A_t, r_t, s_{t+1}) are stored in a large buffer from which mini-batches are sampled to train the Q-network. Due to the promising results in the original DQN paper [126], many variants of DQN [224, 219, 75] have been proposed. For example, Rainbow [75] combines various insights across the years such as double Q-learning [219], distributional Q-learning [16] and prioritised experience replay [152], leading to further performance improvements in the Atari domain.

Limited observation The LSTM network [77], the current state-of-the-art in recurrent neural networks, has been integrated with various reinforcement learners and has compared favourably to other reinforcement learning approaches. RL-LSTM [12] approximates the value function of the above-mentioned Advantage Learning [10] with an LSTM, illustrating comparatively favourable performance compared to Elman networks [112] and the memory-bits [134] approach, on partially observable T-mazes with long-term dependencies of up to 100 time steps as well as a pole-balancing experiment with hidden state variables. Out of these, arguably the most challenging scenarios were solved by Deep Recurrent Q-Networks [69], an extension to the earlier-mentioned DQN which includes an LSTM layer as well; this includes Atari games with flickering pixels and the VizDoom competition related to the first-person shooter game Doom [170, 102, 32]. In the latter, DRQN achieves the highest scores on several metrics and second in other metrics, demonstrating itself to be a scalable algorithm for complex non-Markovian environments. This is not surprising as DRQN includes many desirable features for complex environments, such as experience replay, convolutional layers and an LSTM layer.

Non-episodic environments A positive aspect of value-based methods not based on Monte-Carlo updates, is that bootstrapping of future estimates based on the value-function, as is done in Q-learning and Advantage learning methods, allows to estimate what will happen in the future. However, this bootstrapping is based on the discounted cumulative reward, which implies that events which provide rewards after a long delay do not contribute to learning. This is problematic in non-episodic environments, where delays may stretch far out in the future.

Sparse rewards For sparse-reward environments, RL-LSTM bases its exploration rate on the temporal difference error, a measure of model uncertainty, for a given observation [12], constituting a method for active exploration. Methods using experience replay, such as DQN [126, 69], sample experiences for training in a way that could potentially span the entire lifetime, rather than overfitting on patterns which are not associated with any rewards. This of course assumes that the agent is not stuck in an area without rewards all the time.

Lifelong learning Experience replay is based on a First-In-First-Out mechanism, which can result in catastrophic forgetting, for example, when the buffer only replays the current task. To solve this, distribution matching [84] makes the experience sampling mimic the tasks' distributions, and this was demonstrated on a sequence of randomly ordered autonomous driving tasks, where tasks are conceptually similar, all being defined by a particular goal location, and where the path to take is provided to the learner. Another method to avoid catastrophic forgetting is Elastic Weight Consolidation [95], which modifies the objective function to penalise large deviations from the optimal solution to earlier tasks. Kirkpatrick et al. (2017) applied their system on a randomly ordered task sequence of Atari games, which are characterised by different reward functions and dynamics but where the task could potentially be recognised due to the rich observation. It was shown that a DQN with EWC penalty outperformed DQN without penalty. However, training a single EWC-based DQN for all tasks still resulted in a lower performance than training task-specific DQN learners.

2.4.2 Policy-based learners

The approach with "vanilla policy gradients" [229] has been largely abandoned in reinforcement learning, mainly due to recent performance results. Further, its Monte Carlo style updating which assumes episodic structure is less attractive in long-term continued environments. State-of-the-art policy-based learners include Deterministic Policy Gradient [177], Proximal Policy Optimisation [169], and Trust Region Policy Optimisation [167]. Because they are often implemented as actor-critic learners, they will be explained in the following subsection. However, as an illustration, Trust Region Policy Optimisation was also implemented as a traditional policy-based method in [167], where it was applied to continuous control tasks such as swimming and walking; there it showed a performance improvement compared to a variety of competitive reinforcement learning methods; for full reference list see [88, 203, 136, 135].

Sparse rewards Policy-based learners, and therefore also actor-critic learners, optimise the policy directly and with stochastic policies, this can be favourable for exploration as the degree of stochasticity can be controlled.

Lifelong learning Model-Agnostic Meta-Learning (MAML) [51] is an approach for learning multiple tasks, suitable for gradient descent methods for both supervised and reinforcement learning. The method computes a general parameter vector which serves as an initial point for exploration to more rapidly learn new tasks and as new tasks are seen, this initialisation allows to position the parameter vector more favourably in a region of parameter space where only few gradient updates are required for learning a new task. Although it is not a traditional lifelong reinforcement learner, in the sense that multiple tasks can be incurred at the same time in batch, similar to multi-task learning[1], the method reduces to a lifelong learning approach when the batch size is one, due to the sequential looping over batches. Applying MAML with TRPO as meta-optimiser and REINFORCE as base optimiser allowed good performance on continuous control tasks involving locomotion, and this approach compared favourably to pretraining the network on all tasks and a network with random weights in just a few gradient updates, though it had weak performance compared to adding the task as network input. The Efficient Lifelong Learning Algorithm has been extended to Policy Gradients (PG-ELLA) [2], which they describe as online multi-task learning. PG-ELLA models each tasks control parameters as a linear combination of the components from a sparse shared knowledge base acquired from the previous tasks. They study four types of dynamical system domains, in which the parameters that determine the dynamics system are varied across each of the dimension resulting in 30 tasks for each domain, showing that performance on a new task increases as more tasks are previously seen. The above-mentioned results of MAML and PG-ELLA rely on finding a common representation to all the tasks within a domain and the tasks in a domain all have the same type of goal; therefore, such methods may not be suitable for domains with different goals or reward functions. An extension to PG-ELLA has allowed for additional cross-domain knowledge transfer [25] by using projections from the knowledge base to the task group, however, this requires more knowledge to be given to the learner due to the need to provide the task group index to the learner.

[1]In multi-task learning [15, 30], the emphasis is on offline learning of many tasks in parallel with the aim of finding a single representation that generalises across tasks, while in lifelong learning the emphasis is on the online, sequential learning of an ordered sequence of tasks.

2.4.3 Actor-critic learners

Advantage actor-critic (A2C) and Asynchronous advantage actor-critic (A3C) [125] estimate the discounted cumulative reward as the critic and use Advantage learning as the actor. Another type of actor-critic method is Trust Region Policy Optimization (TRPO) [167] which combines a policy-based optimisation with advantage values. It utilises a surrogate objective based on importance sampling due to which it is possible to sample actions according to a distribution other than that of the unknown newly proposed policy, thereby reducing the variance on the estimated objective. It includes a constraint such that parameter changes are limited to a maximal Kullback-Leibler divergence between the old policy and the proposed new policy. This conservative bias allows TRPO to make changes to the parameters which guarantee monotonic policy improvement. In [168], it was shown to solve 3D-control problems difficult for earlier reinforcement learning algorithms. Proximal Policy Optimisation (PPO) [169] has followed up on TRPO to penalise the parameter change in a continuous way. Since the objective is to maximise $\frac{\mathcal{P}_\theta(A_t|s_t)}{q(A_t|s_t)}\hat{\mathbb{A}}_t$, where $\hat{\mathbb{A}}_t$ is the estimated advantage (cf. Equation 2.4) at time t, $\mathcal{P}_\theta$ the newly proposed policy and q is the sampling distribution, the update will be too large when the ratio $r = \frac{\mathcal{P}_\theta(A_t|s_t)}{q(A_t|s_t)}$ moves away too much from 1. Therefore, PPO proposes instead a clipped surrogative objective, where the ratio r is clipped to $[1-\epsilon, 1+\epsilon]$ but only when the clipping reduces the objective. Thus, when parameter change size is high, there is the potential for a penalty by taking the worst of the clipped and unclipped objective. Competitive results were presented on Atari problems [169], and PPO was the top performer in OpenAI's transfer learning competition in the Sonic Hedgehog domain [130]. PPO has favourable properties compared to A2C and A3C: the use of the Generalised Advantage Estimation [168] reduces variance; the clipping provides a correction for overly large parameter updates, allowing monotonic improvements; the ability to learn with multiple actors is maintained. Whilst the above actor-critic methods are on-policy methods, there is also an off-policy actor-critic approach called Deep Deterministic Policy Gradient (DDPG) [177]. DDPG applies a Deterministic Policy Gradient algorithm to learn a deterministic target policy $\mathcal{P}(A_t|s_t)$ and uses a separate behavioural policy, either deterministic or stochastic, to explore. An example is to sample the behaviour policy from a Gaussian distribution around θ, the target policy parameters. Like DQN, DDPG makes use of a target network and experience replay.

Currently, actor-critic methods have gained in popularity due to these promising results. However, an analysis in [74] shows that many of the comparative results are explained by variability of random seeds, parameter settings, and score normalisations. Further, these methods will often use multiple actors which independently assess the same environment to learn from uncorrelated samples for more stable learning. Compared to experience replay, this has two drawbacks: first, it is not realistic since in real-life learning scenarios it would require multiple physical agents at the same time; second, not all comparisons to other methods may be fair, since multiple actors gather more experience in the same amount of time steps. With these cautionary remarks in place, these methods have been successful in various types of video games, but also in continuous control problems, thereby distinguishing it from value-based learners.

Limited observation A3C [125] has been integrated with an LSTM layer outperforming feedforward versions of A3C and several feedforward DQN variants [126, 152, 128, 224], on a variety of Atari games with limited time dependencies. RDPG [72] similarly extends DDPG with an LSTM network, enabling it to learn better than the feedforward version of DDPG on a Morris water maze. PPO has yielded impressive results when integrated with LSTM [73], demonstrating robust locomotion behaviours such as crouching, jumping, and turning by subjecting the learner to more diverse environments.

Sparse rewards Stochastic policy-based and actor-critic methods can additionally incorporate an entropy regularisation component to penalise solutions with high determinism [125, 168]. This allows to explore the environment without getting stuck in sub-optimal near-deterministic policies. An intrinsically motivated system based on exploration counts was shown to improve performance on difficult Atari games, including Montezuma's revenge where rewards are highly sparse [17]. Further, a large scale study compared various surprise-based intrinsic motivation mechanisms on a large variety of problems such as Atari games, Mario, and Roboschool problems [28]; this study showed that a dynamics-based surprise reward, which motivates the agent to explore observations with large prediction errors, yields both performance and transfer benefits to PPO. To improve exploration, a meta-learning approach applied policy gradient methods to train DDPG's behaviour policy [235]. This was then demonstrated on continuous control tasks, outperforming the non-adaptive strategy for exploration usually used in DDPG. A downside of this approach is that the short-term objective used to train the meta-learner may not be applicable to long-term environments.

Lifelong learning Similar to Model-Agnostic Meta-Learning, the meta-critic approach in [198] also investigates a model-agnostic approach with tasks being incurred in batches, which is convenient but not realistic for online learning in long-term environments. There, the implementation is based on an actor-critic model which parametrises the critic with the current task. This method outperformed Model-Agnostic Meta-Learning, more rapidly learning new tasks in the cartpole control domain. Further, there have been positive results on transfer learning when using PPO as a base reinforcement learner. In two studies, PPO was shown to transfer from one game level to another [28, 130]. In another study, it was shown that subjecting PPO to a curriculum of sorts can improve robustness; by letting an agent perform the same locomotion task but across diverse terrains, presented randomly in a variety of ways involving mixing within episode and

Table 2.1: *Features of state-of-the-art reinforcement learning methods to deal with challenges related to long-term learning in unknown environments. + and - indicate positive and negative, whilst an empty space means no relevant feature. Bold face indicates a special significance in the coming chapters.*

Type	**Examples**	**Limited observation**	**Sparse rewards**	**Non-episodic**	**Meta-learning**	**Lifelong learning**
Value-based	DQN	DRQN +	experience replay +	bootstrapping +; **discounting -**	**fixed exploration schedule -**	distribution matching + EWC + **task-specific policy better -**
Policy-based	TRPO		optimises policy itself + online updates -	bootstrapping +; discounting -		
	MAML		optimises policy itself + online updates -	Monte Carlo -	*	learn task-specific policies from initialisation policy + task distribution not realistic -
	PG-ELLA	MDP -	optimises policy itself + online updates -	Monte Carlo -		shared basis for transfer +
Actor-critic	A2C/A3C	LSTM layer +	optimises policy itself + online updates -	bootstrapping + ; discounting - multiple actors -		
	DDPG	RDPG +	optimises policy itself + experience replay + fixed exploration schedule -	bootstrapping +; discounting - multiple actors -	meta-policy gradient : learning of behaviour policy + short-term gradient descent -	
	PPO	LSTM layer+	optimises policy itself + online updates - intrinsic motivation +	bootstrapping +; discounting - multiple actors -		**transfer learning results +**
Universalist	MC-AIXI	context tree + scalability -	fixed exploration schedule -	limited horizon -		
	ALS	working memory + active perception + inefficient and unadaptive perception -	long-term objective +	repeatable simulations -	search over program space +	transfer learning results + assumption about reward velocity -
	IS	working memory + **active perception +** **inefficient and unadaptive perception -**	optimises policy itself + **long-term objective +** Dual Brain +	long-term objective +	unlimited meta-levels + **optimising learning itself +**	transfer learning results + **requires curriculum -** **assumption about reward velocity -**

Note (*): the Model-Agnostic Meta-Learning defines meta-learning in terms of learning of different tasks. In this work, meta-learning is defined as the ability to learn how to modify aspects of the learning algorithm.

between episodes, robust locomotion behaviours emerged after training [73]. In sum, the positive generalisation results suggest a role for PPO as a suitable base-learner for lifelong learning.

2.5 Universalist reinforcement learning methods

While deep neural networks benefit from the capacity of neural networks, the way those neural networks are trained is in itself not part of the learning, and this limits the extent to which the learning method works on unknown environments. True meta-learning is best represented by the universalist RL methods, which perform a general search in program space using a single meta-algorithm. Due to the possibility of learning how to compute arbitrary learning algorithms, the approach is promising in principle, although not as widely explored as deep reinforcement learning.

The Goedel Machine (GM) [160, 162] is an instance of what is called a self-modifying policy (SMP), meaning it is able to learn how to make modifications to its own policy, and which represents Recursive Self-Improvement (RSI) [236, 193], an approach in which the original algorithm may be completely replaced by a novel improved algorithm. The GM proposes modifications to itself, namely its axioms, its proof search, or its policy, but proposals are only accepted if they are mathematically proven to improve expected utility. Despite its optimality in some sense, the GM is not the most practical approach: (a) due to apparent inconsistencies in the real-world and the limitations to provability by Goedel's incompleteness theorems, some statements will be unable to be proven and the self-modification system may be completely inert; (b) no working implementation exists though some ideas have been proposed in [191, 192]. Another approach to obtain a universal learner is, instead of shifting a learner's inductive bias, to use an inductive bias suitable for many problems. AIXI [81] maintains a Bayesian mixture model over possible environments, weighting the simplest environments more than the complex environments according to the length of the programs to generate them. This respects philosophical principles such as Occam's razor, which states that the most simple explanation should be preferred, and Epicurus' principle of noncontradiction, which states that all hypotheses consistent with the data should be maintained. Unfortunately, the approach is not practical, as it relies on the uncomputable Kolmogorov complexity of the environments. A practical approximation called MC-AIXI was proposed in [222] but this suffers from the limitations of tree-based learners, which have limited scalability and limited horizon. Additionally, recent theoretical research has invalidated AIXI's previous universality claims [106].

So far, the most practical universalist RL methods are based on the Success Story Algorithm (SSA) [166]. SSA performs a back-tracking over modifications to the meta-program until the system only maintains those modification sequences that, starting from their introduction, result in lifetime reward acceleration. Various implementations of SSA have been proposed for meta-learning. Adaptive Levin Search [166] performs Levin Search, searching candidate programs according to their Levin Complexity, increases the probabilities of those instructions that lead to the correct solution of the problem, and then evaluates those probability changes with SSA. This increased performance on a big episodic partially observable maze and transfer learning in a maze with varying goal-sets, when compared to either Levin Search or Adaptive Levin Search without SSA. Another implementation, called Incremental Self-improvement (IS) [166, 157], similarly modifies instruction probabilities and evaluates them with SSA, but builds algorithmic solutions based on self-referential instructions to inspect and modify the probabilities of the instructions. Similar to the Goedel Machine, the method is part of the wider class of Self-Modifying Policies (SMPs), which due to encoding self-modifications into the action set have an unlimited number of meta-levels. The method treats computational procedures similar to external actions on actuators, and this allows resource-bounded rationality [145, 156]. Further, it allows the coordination of attentional, sensory, and learning operations efficiently with the external actions; therefore, such systems can also be used for a general form of active perception. So far IS has been demonstrated on non-episodic environments [157], multi-agent systems [164, 241], partially observable environments [164, 241, 157], noisy environments [240], as well as continual learning, solving problems of increasing complexity presented sequentially across the lifetime [166]. In the past, this approach has compared favourably over various versions of Q-learning, although it is not clear how Incremental Self-improvement would compare to modern versions of Q-learning such as DQN or DRQN.

Limited observation On a variety of partially observable environments,including a challenging partially observable pacman problem, MC-AIXI achieves higher performance when compared to tree-based learners U-Tree [118] and Active-LZ [47]. Like other tree-based learners, a downside of MC-AIXI is the scalability when the time dependencies and observation or action spaces grow larger. Some Incremental Self-improvement systems allow reading and writing information on a working memory [157] to provide general perception and memorisation, but this has limited efficiency due to modifying only a single variable at a time. Other implementations include active perception instructions [164] to allow checking for user-specified objects at times chosen by the learner. This has at least three limitations: (i) it is assumed the objects that may appear are known in advance; (ii) the exact implementation of these instructions is not provided; (iii) their categorisation mechanism cannot adapt to environment changes.

Sparse rewards and non-episodic environments MC-AIXI has a limited horizon which makes it unsuitable for non-episodic environments with sparse rewards. Adaptive Levin Search tests programs on a repeatable simulation, which is not realistic for on-line learning in non-episodic environments. In contrast, Incremental Self-improvement allows evaluation at self-chosen time-points, by learning when to perform an instruction which initiates the Success Story Algorithm. SSA then repeatedly performs evaluation of past policies by back-tracking over previous self-modifications and calculating whether or not they improve the lifetime reward velocity. Because the evaluation spans the entire lifetime, this makes Incremental Self-improvement suitable for non-episodic and sparse reward environments such as the maze setting in [157]. Incremental Self-improvement has also been extended for sparse reward environments in the Dual Brain framework [158]. This framework utilises a combination of two interacting Incremental Self-improvement learners, each of which obtain surprise rewards whenever one predicts a correct outcome and the other predicts a false outcome. Illustrative of the pros and cons of intrinsic motivation, the method was able to gain prediction capabilities even when there was no feedback from the environment, but completely ignored the external rewards in some runs.

Meta-learning In most reinforcement learning systems, there is an assumption of some knowledge about the time scales of a problem, as is visible for example in fixed updating frequencies, fixed evaluation frequencies, and exploration schedules which for example decrease the exploration rate from time $t = 0$ to some later time, e.g., $t = 10^6$. Further, they assume a fixed learning algorithm with no modifications. Another key assumption is that, according to the agent-environment interface, the learner's algorithmic procedures do not consume any time. Incremental Self-improvement does not need to make these assumptions, and instead constructs algorithmic procedures based on elementary instructions. This property of Incremental Self-improvement also allows a form of real-time reinforcement learning. Unlike existing approaches to real-time reinforcement learning [76, 222, 178, 200, 9], Incremental Self-improvement learns how much time should be spent doing which learning operations.

Lifelong learning In one study [166], Incremental Self-improvement has been successfully applied to lifelong learning scenarios, solving related mathematical functions of increasing complexity, using a learning schedule which initially provided easy tasks frequently and gradually increased the frequency difficult tasks. Due to the well-designed learning curriculum, the study yields only limited conclusions about lifelong learning in unknown environments, where: (a) tasks can be presented in a more random order; (b) tasks may not relate to each other; and (c) discontinuous reward function changes may occur.

2.6 Other lifelong reinforcement learning approaches

Many other reinforcement learning methods are fairly similar in assumptions and mechanisms when compared to traditional and deep reinforcement learning. This section reviews those methods which are specifically proposed for improved lifelong learning.

2.6.1 Hierarchical approaches

A variety of hierarchical reinforcement learning approaches break up reinforcement learning in smaller problems, enabling to find routines which are re-usable across different subtasks to provide increased transfer learning. SKILLS [209] defines sub-policies based on a partition of the state space, learning those skills shared across tasks. The Options approach [201] uses special hierarchical actions called options, which are initiated based on particular conditions, then define a particular state-action policy to follow for that option, and then terminate in a particular set of states. The Options approach was extended in [26] for Option discovery, utilising a two-phase structure to first extract options in various MDPs and later combine them to learn a policy using the options. A hierarchical approach called Hierarchical Deep Reinforcement Learning Network (H-DRLN) [207] defines a policy over primitive actions, first-order skills pre-trained on simple tasks, and second-order skills which combine first-order skills based on policy distillation [147], a method to transfer knowledge from a teacher to a student model. Compared to other approaches in this category, H-DRLN can be applied in complex state spaces, as demonstrated by the use in MineCraft.

A downside of these approaches is that they are typically assuming a two-phase structure, in which first re-usable subroutines are learned, and then the hierarchical model over these routines is learned. This means that the subroutines are not learned online, limiting their adaptivity. Also these approaches make the Markov assumption, which prevents learning with limited observation. Apart from the H-DRLN approach, the above-mentioned approaches have limited scalability. Also some of these approaches, such as the ones mentioned in Konidaris et al. [96], require special domain knowledge.

2.6.2 Constructive approaches

Various authors have explored an incremental approach to build skills of increasing complexity, by expanding the number of units. CHILD [140] demonstrated an approach using a constructive neural network for Q-learning, increasing the historical context to take into account for decision making to solve increasingly difficult problems. Progressive neural networks [146], which add a new sub-network for each new task while adding lateral connections to previous tasks, were previously applied on sequences of Markovian environments including Atari games, pong and 3D mazes (each class a separate lifelong learning scenario). This includes sequences containing tasks with orthogonal properties, although in many cases detecting the task that is being solved can be inferred from the observation. In such scenarios, the progressive networks strategy has been shown to increase positive and reduce negative transfer. Beyond the above, literature on constructive networks for online reinforcement learning is sparse, and the few approaches that do [218] are not tailored to the lifelong learning scenario.

A downside of this approach is that not all learning is fundamentally incremental, and that the complexity of these methods grows over time. Approaches based on constructive networks also add neurons based on relatively heuristic hand-crafted criteria; a challenge is to allow adaptive learning of network construction.

2.7 Summary

Neural networks provide a solid foundation for adaptive perception. Recurrent neural networks have a high capacity, as they can represent complex dynamical systems, but are difficult to train. With many free parameters, they need high quantity of representative data. When there is only a single sequential lifetime, such as in real-life robotic systems, methods such as evolutionary algorithms cannot be applied as they require repeatable trials and controlled conditions. Gradient descent algorithms get stuck in local optima difficult to escape, which limits the type of functions they can learn.

Neural networks have been integrated with reinforcement learning in the field of deep reinforcement learning. Value-based learners and actor-critic methods have been especially popular in deep reinforcement learning literature. Due to the use of target networks and experience replay which allow more stable learning in long-term environments with sparse rewards, Deep Q-Networks and variants thereof have been applied successfully to game playing environments. One variant is Deep Recurrent Q-Networks, which combines DQN with Long Short-Term Memory to obtain state-of-the-art performance in partially observable environments. Value-based learners are limited in lifelong learning, as learning task-specific DQN policies outperformed the lifelong learning DQN with Elastic Weight Consolidation. For actor-critic methods, Proximal Policy Optimisation has generally yielded the best performance, and has favourable transfer learning properties. Similar to the DQN system, PPO has also been extended to include Long Short-Term Memory networks for partially observable environments.

Although meta-learning, learning how to learn, has been studied to improve adaptivity in deep reinforcement learning, such studies have been limited to systems with only a single meta-level, where one meta-optimiser optimises a small part of a subordinate policy. In comparison, the universalist tradition of AGI better represents the true meaning of meta-learning, allowing multiple meta-levels by considering the learning algorithm as an input to itself. Incremental Self-improvement is the most practical in this approach and using its long-term utility function in principle allows learning regularities in long-term environments; however, (a) it is unknown how it compares to modern deep reinforcement learning systems; (b) its instructions for perceiving the world are unadaptive and unrealistic; (c) its lifelong learning results depend on a well-designed curriculum, rather than an unknown environment.

Chapter 3

Long-term adaptivity to alleviate and shift inductive bias

A key property introduced in Chapter 2 is capacity or expressiveness, a quantitative metric for how many functions a learning system can represent. There, it was discussed how recurrent neural networks can in principle represent all open dynamical systems but how in practice this can be difficult to achieve due to the limitations of the training method and the data provided to the training method.

Inductive bias [124] is a more qualitative concept which refers to what types of functions a learning system can represent and why it can or cannot represent them. It refers to the set of assumptions which determine how to generalise observed data to unobserved data. When solving an unknown long-term environment, this becomes incredibly important as the data distribution of the future is unknown and, in reinforcement learning, may be dependent on the actions of the agent. As discussed in the literature review, a bias for a limited number of functions, corresponding to a low capacity, would be beneficial for rapid learning, but comes with the risk that no valid solution can be expressed. Therefore, ideally, the learner is open to a wide set of functions, corresponding to a weak bias, but has long-term adaptivity of representation and learning to shift the bias for more efficient learning.

In this context, this chapter presents the main research methodology used in the thesis. First, the chapter identifies the bias in deep reinforcement learning systems. The motivation behind the case studies, consisting of a continuing single-task scenario and a lifelong learning scenario, is presented. Finally, the methods for reinforcement learning with long-term adaptivity are proposed, with emphasis on the key concepts rather than the implementation level.

3.1 The inductive bias of deep reinforcement learning systems

Bias may be present in the representation itself or in the algorithmic procedures. Below is a list of biases that make it difficult to learn in long-term unknown environments:

- Limited memory: in general non-Markovian environments, the system usually cannot memorise all previous events that happened in the causal chain. This makes it risky to simply try to memorise all patterns, because observations and decisions made much earlier might affect which decision should be taken now.
- Architectural constraints: the fixed hidden layer sizes may not be suitable for the environment. Too many neurons come at the risk of overfitting, while too few come at the risk of not being to learn complex patterns.
- Limited evaluation horizon/Short-term objectives: evaluation with the cumulative discounted reward inherently means that learner's decision do not get a complete picture of the environment, as rewarding or punishing events in the distant future are not accounted for. Such learners therefore implicitly assume the distant past is not relevant to making decisions.
- Locality of the search space: methods of the gradient descent type are most widely used in deep reinforcement learning, but a problem is that they simply descend into the most promising regions. The assumption of smooth, continuous objective landscapes with reliable progress indicators is not always true, and therefore such methods risk getting stuck in a local optimum.
- Untargeted exploration: when the environment is unknown and incurs only sparse rewards, exploration is important to find the higher-density reward regions and learn how to consistently reach them. Without any idea of how to best explore the environment, deep reinforcement learners can potentially get lost forever, as the best learned actions might be preventing to reach the good locations. The assumption that an unadaptive exploration mechanism, such as epsilon-greedy, or adaptive exploration based on a short-term objective, such as meta-policy gradient [235], will lead to exploring all the possible patterns is not always true, especially when rewards are sparse.
- Unadaptive learning: whilst reinforcement learners are inherently adaptive, the learning mechanisms themselves are not adaptive. For example, although Q-learning updates adapt the value of state-action pairs over time, the learning itself is static since the same type of update is being applied on the same tabular or neural network representation. This makes it impossible to obtain a good performance if, due to one of the reasons mentioned above, the learning algorithm or representation is not suited to the environment.

- Catastrophic forgetting and negative transfer: learning one task may actually hamper the performance on other tasks. With features of the tasks at hand unknown and the task being difficult to classify, deep reinforcement learners struggle to solve lifelong learning scenarios.

The above biases limit the possible environments which deep reinforcement learners can accurately learn. This is not necessarily bad: if the bias is *correct*, then the deep reinforcement learner is able to correctly learn the patterns for the environment it encounters, whilst avoiding an unnecessarily large search space; such learners are said to have a high expressive efficiency. If the bias is *false* in a strict sense, for example when some patterns occurring in the environment cannot be expressed by the representation, then the deep reinforcement learner will never be able to correctly learn the environment.

It is also possible that, although it is theoretically possible to learn the pattern, in practice the corret pattern is never learned because of the practical limitations of the training algorithm. This can be particularly well illustrated for gradient descent methods, as shown in Figure 3.1. A gradient descent search effectively defines a practical search space: because it follows the gradient and it cannot reach certain regions of parameter space when the objective landscape is deceptive. Lifelong learning also presents similar challenges: when learning different problems, performance on earlier tasks may be hampered because the agent learns a parameter set that is somewhere in between the different tasks' optimal parameter settings.

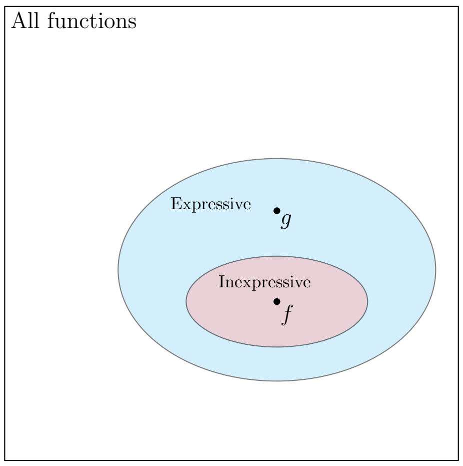
Meta-learning is one potential approach to mitigate this problem. By selecting different representations centered around a different region in parameter space or even changing the dynamics of the search itself, meta-learning can effectively change the practical search space. As such, it can shift bias based on experience.

3.2 Case studies

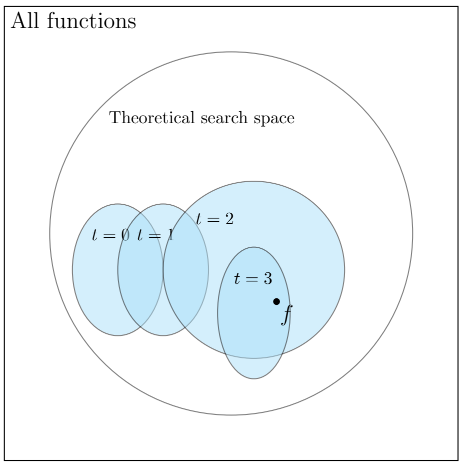
Due to their relevance for solving long-term unknown environments, the sources of bias mentioned above inspire the investigations into two case studies.

3.2.1 Case 1: long-term learning with limited knowledge and sparse rewards

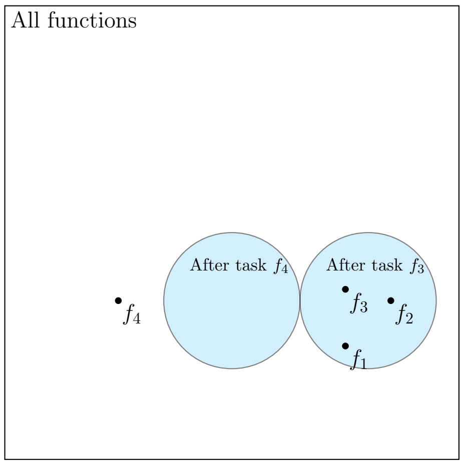
As a first case study, Chapter 4 considers a maze problem with several challenging characteristics. The observation is limited, leading to partial observabilty and the need to remember the recent history of events. The environment appears as non-episodic: the learners are not aware of any terminal states and the designer has no control over the environment in the sense that when learners fail to reach the goal for prolonged amount



(a) Trade-off between capacity and expressive efficiency



(b) Meta-learning



(c) Lifelong learning

Figure 3.1: *Illustration of inductive bias.* ***(a):*** *Sizes of the ellipses denote the theoretical search space of the learners. The 'Inexpressive' learner has a high capacity which means it can represent a small set of functions and therefore has a representational bias. This makes it more efficient at learning a particular problem f for which its bias is correct, but can never learn the function g for which its bias is incorrect. The 'Expressive' learner can learn both f and g in principle, but because its search space is larger it may not always find the solution in practice.* ***(b):*** *Meta-learning with a single meta-level can be seen as a search for the best* practical search space*, here shown in blue at different time steps, by adjusting its shape and location in the space of all functions, in the search of the best procedure which allows it to best learn the optimal function f. Meta-learning with two meta-levels can then be understood as the search for the search in practical search space.* ***(c):*** *In lifelong learning scenarios, a key challenge is to search for solutions to new tasks efficiently without losing solutions on old tasks; two common strategies are using separate policies for each task guided by an initialisation and learning a single policy which captures aspects of all tasks. When a new task is presented that has a different optimal set of parameters, these approaches reposition their policy and therefore the practical search space to better capture the range of tasks. As a consequence, performance on subsequent tasks which are similar to f_1, f_2, and f_3 may now be reduced.*

of times there is no means to let them know this. This non-episodic property makes the evaluation horizon span out far in the future and makes it possible to get stuck in a location far away from the goal. Due to the sparse feedback, with rewards only being received at the goal location, the learner must explore the environment intelligently to avoid getting stuck.

3.2.2 Case 2: lifelong learning

As a second case study, Chapters 5 and 6 investigate a lifelong learning scenario. Unlike usual reinforcement learning scenarios, the environment will, at different points of the learner's lifetime, present a new task to be solved. In traditional reinforcement learning, based on Markovian Decision processes, there is a single reward function $r(s_t, A_t)$ in which each state-action pair is associated with a real-valued reward, and the state transitions follow a Markovian transition probability model $\Pr(s_{t+1}|s_t, A_t)$. In the lifelong learning scenario, different tasks have completely different non-Markovian dynamics, as well as different reward functions. The learner then must adapt to this task, transferring its knowledge from previously seen tasks, or, if the task was already learned, remember the relevant skills used to solve it. The tasks are presented in unknown order, and, due to the partial observability, simply observing the environment does not yield any information about the task at hand. This makes it difficult to transfer knowledge selectively and to prevent updates from erasing previously learned knowledge; therefore, effects of negative transfer and catastrophic forgetting pose a particularly strong challenge in this environment. As in the first case study, the partial observability in itself poses a challenge, as the learner needs to remember many previous time steps.

3.3 Long-term adaptivity

To address these challenging case studies in a task-agnostic manner, the thesis investigates the role of long-term adaptivity to solve and diagnose issues related to the inductive bias of deep reinforcement learners. Various aspects of long-term adaptivity are included into the thesis, in the sense that: (i) long-term objective functions are included to consider a lengthy time-span such as the entire lifetime; (ii) long-term adaptations to learning itself, based on many prior evaluations; and (iii) selective use of representation which spans a longer time-frame than is usually the case. A novel learning type called "active adaptive perception" incorporates (i) and (ii). A different approach called "learning with multiple policies" incorporates (iii) as a feature and further serves as an analysis tool for lifelong learning, with the aim of finding suitable training methods and objective functions for long-term environments.

3.3.1 Shifting bias with Active Adaptive Perception

The issue of inductive bias is highly relevant for autonomous systems as they must make sense of the world using their perception. Perception is commonly defined as the ability to interpret sensory information to represent the external world [149]. Bias in perception may lead learners to fail, either completely or for particular spatio-temporal patterns, when they are subjected to unknown environments for which they are not specialised. Even though the representation using recurrent neural networks may be sufficient to capture all the relevant patterns in theory, in practice this will not always be true, due to the bullet points in Section 3.1.

Among the points raised, "Unadaptive learning" is the limitation that unadaptive learning methods may find themselves unable to solve a given environment with no ability to make according adjustments. To remedy this limitation, here it is proposed that allowing agents to flexibly use and modify their perception may provide improved performance when the agents are granted a sufficiently long lifetime to learn how to learn. Interestingly, this bullet point is relevant for many other bullet points as well: limited memory, architectural constraints, locality of the search space, and untargeted exploration are key domains that may be improved by allowing more autonomously generated routines for learning and behaviour.

To further explore this reasoning, Chapters 4 and 6 explore a novel learning type, further called *active adaptive perception*, defined based on two conjunctive characterisations.

A first characterisation (C_1) is as *an active form of adaptive perception*. Adaptive perception makes long-term adjustments to the perceptual system based on the environment's feedback; to be an active adaptive perceiver then means to be able to decide how and when to modify the perceptual system, based on environmental demands. Therefore, active adaptive perceivers include a significant meta-learning component to learn how to improve their adaptive perception.

A second characterisation (C_2) is *active perception*, using Bajcsy's definition "an agent is an active perceiver if it knows why it wishes to sense, and then chooses what to perceive, and determines how, when and where to achieve that perception" [11]. This broad definition of active perception includes but is not limited to actively choosing when and how to reposition the sensors, interpreting sensory information, or predicting incoming sensory information.

By combining these two characterisations, active adaptive perception allows learning how to modify and apply perception. Due to the large scope of active adaptive perception, and the inherent complexity in analysing such flexible behaviours, the thesis will focus on simple cases, by investigating systems which learn:

- which experiences are useful to pursue as goals (C_1)

- which parts of a neural networks should be modified (C_1)
- which learning rate is suitable in which situations (C_1);
- which exploration rate is suitable in which situations or for which goals (C_1-C_2 interaction);
- when and how to interpet sensory information (C_2), by deciding
  - how frequently to rely on a neural network representation versus a probabilistic instruction-based representation;
  - for which goals to rely on a neural network representation;

A generic architecture To achieve active adaptive perception, the thesis proposes a generic architecture illustrated in Figure 3.2. The architecture consists of four basic components: an instruction module, an evaluation module, a working memory and a perception module. On a conceptual level, the difference with a traditional self-modifying policy such as Incremental Self-improvement is the introduction of a *perception module*, a sub-symbolic system that provides a selective use of representation and efficient pattern recognition to guide the subsequent instruction cycles. The architecture serves as an abstract template for learners with active adaptive perception; the way the architecture is implemented may alter efficiency but not the property of active adaptive perception. To emphasise this, the specifics of its implementation are left for the following chapters; this subsection serves to explain the generic properties of the modules, and to point to the connection of the perception module to Chapter 4 and the evaluation module to Chapter 6.

The **instruction module** organises the interactions with the environment but also with the different modules of the architecture by utilising a user-defined instruction set $\mathcal{A}$, a set of operations which includes external actions which involve interacting with the environment, e.g. moving one step north, grabbing an object, or applying sensory mechanisms; and internal operations to enable memory, learning and inference. The mechanism of the instruction module is to continuously generate instructions based on the current instruction module parameters $\mathcal{P}$ and a set of working memory variables. Throughout the thesis, the instruction module will be based on a probability matrix in which rows represent the internal state of the learner whilst the columns represent the instruction used. The instructions, similar to functions used in programming languages, take several arguments as inputs and then perform some computations based on these arguments.

The **perception module** is a modifiable sub-symbolic module whose function is to advise the instruction module, using special instructions relevant for active adaptive perception. It supplies bottom-up perception to the architecture by analysing sensory

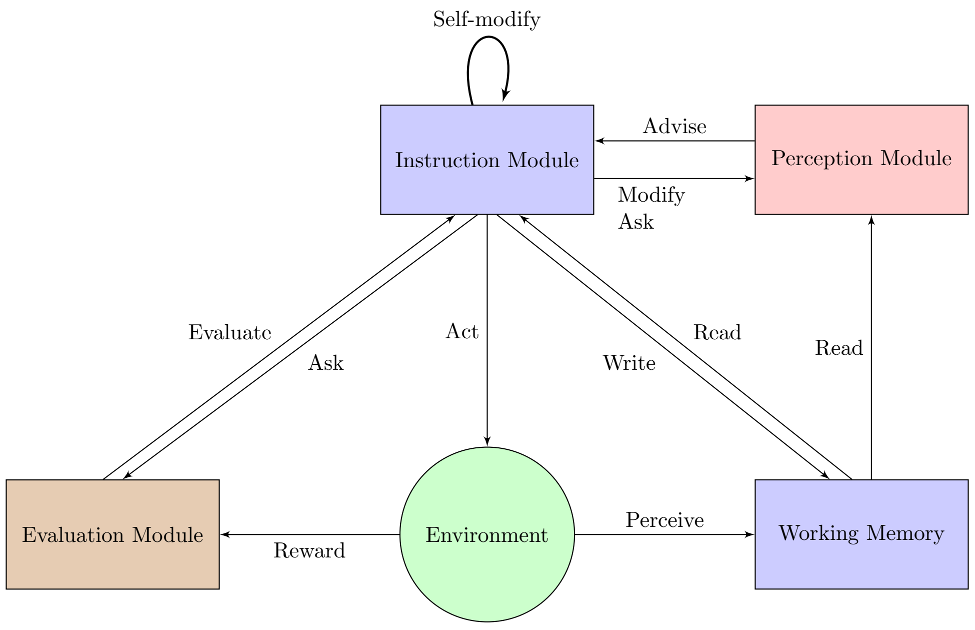


Figure 3.2: *Block diagram of the generic architecture. This architecture for active adaptive perception serves as the basis for the next two chapters. The Perception Module (red color), investigated especially in Chapter 4, is used to inputs and historical variables stored in the working memory. The Evaluation Module (brown color) is the topic of Chapter 6 where variants of the Success Story Algorithm are investigated.*

inputs and working memory variables in terms of high-level concepts to help decision-making. For example, successive layers of a neural network may process elementary visual stimuli such as edges into shapes and objects, and a configuration of free and blocked spaces in a maze may be processed in terms of a narrow corridor or a wide area. By interacting with the working memory and the instruction generation, it can influence the decisions made by the instruction module, using *perceptual advice* instructions. The architecture and the parameters of the perception module are subject to long-lasting modifications when the instruction module calls special *perceptual modification* instructions. By learning when to use which perceptual advice and perceptual modification instruction, various strategies for utilising and training the perception module will emerge from experience, leading to active adaptive perception. Chapter 4 investigates this module in particular, to illustrate a range of possible perceptual advice and perceptual modification instructions.

The **evaluation module** uses a long-term objective function to determine whether changes to the instruction module and the perception module are beneficial for long-term reward intake. The evaluation module is therefore a key factor to facilitate adaptivity. Before explaining roughly the ideas behind this module, consider the Section 3.1 bullet point of "limited evaluation horizon". Traditional reinforcement learning systems consider the future discounted reward as the main objective. For meta-learning, as well for learning in long-term environments with no clearly defined episode boundaries, this is not suitable. Instead, the evaluation module in Chapter 4 will be based on the Success Story Algorithm. The Success Story Algorithm evaluates *self-modifications*, little chunks of information which when introduced modify the policy of the learner whilst when removed revert the learner back to the policy from before the modification, similar to a version control system. To keep of track of all the self-modifications, the algorithm makes use of a data structure called the stack, further denoted as $\mathscr{S}$. Table 3.1 shows the list of operators that will be used throughout Chapter 4 and Chapter 6. To account for delayed effects of self-modifications, the stack is evaluated in terms of *self-modification sequences*. This is done based on the Success Story Criterion, which asserts whether or not the lifetime reward velocity has speed up since the self-modification has been introduced. If this is not true, the self-modification sequence is removed, and the policy from before the removed self-modifications is restored. The evaluation module is of critical importance in this thesis; in Chapter 6, the Success Story Algorithm is extended to better be able to evaluate self-modification sequences in lifelong learning environments, especially when different tasks have completely different reward profiles.

As a last module of the generic architecture, the **working memory** is used to store and manipulate information from various parts of the learning structure as well as the environment. Variables in the working memory are updated regularly or at self-chosen times by the instruction module and the environment. The working memory provides other modules with historic information, contributing to non-Markovian learning and decision-making.

Algorithmic Overview Now that the meaning of the various modules has been explained, the order of the various interactions between them can be further clarified. In general it will always be assumed that a learner, $\mathcal{L}$, is put in an environment, $\mathcal{E}$, with the aim of maximising a long-term objective such as the lifetime cumulative reward $\sum_{t=0}^{\infty} r_t$. $\mathcal{L}$ seeks out rewards by outputting an instruction a from a user-defined set of instructions $\mathcal{A}$. The learner's instruction cycle works as follows. The agent, with its D sensors, receives an observation $o \in \mathcal{O} \subset \mathbb{R}^D$ from $\mathcal{E}$ and writes it directly to the working memory elements reserved for observation. Based on a fixed subset of working memory cells, not necessarily the same as observation elements, the instruction module, based on the instruction module parameters $\mathcal{P}$, generates an instruction $A \in \mathcal{A}$ and its arguments $a_1, \dots, a_N$, if that instruction has any. The execution of A results in interactions between modules (internal actions) or between $\mathcal{L}$ and $\mathcal{E}$ (external actions $A \in \mathcal{A}^E$); types of interactions are described in the following paragraph. As in reinforcement learning, a critic in the environment $\mathcal{E}$ sends a reward $r \in \mathbb{R}$ after an external action is taken. The cycle ends after $\mathcal{L}$ has processed the reward in the evaluation module.

Interactions between modules are encoded in the instructions. This flexibility allows true meta-learning, as computational elements are treated in exactly the same way as

Table 3.1: *Operators on the stack $\mathscr{S}$ and data stored in stack-entries $e \in \mathscr{S}$. The stack is a convenient datastructure to enable back-tracking across incremental self-modifications, and this is used as part of the implementation of the evaluation module.*

Type	**Operator**	**Meaning**
Stack operators	$\mathscr{S}$	the stack containing all valid modifications
	$\mathscr{S}[i]$	get the i'th element in $\mathscr{S}$
	$\mathscr{S}[sp]$	the top-most element of the stack
	$\mathscr{S}.push(e)$	add a new entry to the top of $\mathscr{S}$
	$\mathscr{S}.pop()$	remove $\mathscr{S}[sp]$
Stack-entry data	e or $e \in \mathscr{S}$	a single entry in the stack, with data to restore the learner to its settings before the modification were made
	$e.first$	pointer to the first entry in the same self-modification sequence as e
	$e.t$	the time at which e was pushed
	$e.R$	the cumulative reward at time t, $R(t)$
	$e.\mathcal{X}$	the data to recover the instruction module or perception module as it was before e was pushed.
	$e.address$	pointer to the location where the instruction module or perception module is modified; helps to avoid storing the entire policy
Firsts subset	$Firsts$ or $Firsts \subset \mathscr{S}$	a convenient mathematical notation, $Firsts = \{e \in \mathscr{S} \mid e = \mathscr{S}[e.first]\}$ includes only those entries which started a self-modification sequence; stack-operators can be applied analogously.
	$Firsts.top()$	get the top-most element of $Firsts$; is equivalent to getting $i \leftarrow \mathscr{S}[sp].first$, where sp points to the top index of $\mathscr{S}$, and then obtaining the entry $e = \mathscr{S}[i]$.
	$Firsts.secondtop()$	get the second top-most element of $Firsts$; is equivalent to getting $i \leftarrow \mathscr{S}[\mathscr{S}[sp].first - 1].first$, where sp points to the top index of $\mathscr{S}$, and then obtaining the entry $e = \mathscr{S}[i]$.

interactions with the environment. The following types of instructions are the minimal requirement for active adaptive perception:

- External actions ($A \in \mathcal{A}^E$): interact with $\mathcal{E}$ to obtain rewards
- Self-modification ($A \in \mathcal{A}^{IM}$): modify the instruction module parameters $\mathcal{P}$.
- Working memory manipulation ($A \in \mathcal{A}^{WM}$): change the working memory based on the sensory inputs and the current working memory
- Perceptual advice ($A \in \mathcal{A}^{PM}$): based on the current working memory a part of the perception module computes outputs which influence instruction generation.

- Perceptual modification ($A \in \mathcal{A}^{PM}$): modify the perception module's parameters, changing the way perceptions form and interact with the instruction module.
- Evaluation ($A \in \mathcal{A}^{EM}$): call the evaluation module to evaluate changes to the instruction module and the perception module

When all instructions are regularly used, this results in the following learning steps:

1. The learner starts with a minimally biased instruction and perception module, where the instruction probabilities are uniform and the perception module's parameters are randomly initialised;
2. Intermittently, changes to instruction generation and the perceptual processes are made;
3. Once the evaluation module is called, the changes are accepted or rejected based on evidence of their contribution to reward intake;
4. The instruction module thus learns when to execute the instructions but also how; the meaning of instructions is optimised as the arguments supplied to the instruction is changed.
5. This leads to optimisation of the interaction between the various modules. One of the consequences is perceptual learning, which may be divided into two processes:
    - long-term parameter changes: similar to traditional sub-symbolic learners, the best parameters for a given perceptual operation are learned.
    - active adaptive perception: learning how to modify and use the perception module.

Blending the universalist and the sub-symbolic tradition The above architecture is inspired by two different AGI traditions discussed in Section 1.2.3, namely the sub-symbolic tradition and the universalist tradition. The sub-symbolic approach, of which deep learning is a major part, is particularly strong in recognising complex spatio-temporal patterns. The universalist approach by contrast prides itself on the ability to find good or optimal solutions in vast algorithmic search spaces. The underlying rationale behind the generic architecture is that a universalist algorithm searches the space of possible programs for learning, constructing and applying one or more sub-symbolic learners which then supply perception to the system.

This works as follows. The instruction module, a universalist meta-algorithm, utilises elementary instructions to construct perceptual modification algorithms and to selectively apply the perception module. In contrast to deep reinforcement learners, which use a fixed update rule, a deep neural network or other sub-symbolic representation can be modified in this manner by learning from long-term experience how to combine elementary update steps included in specialised instructions – a process called perceptual modification. Similarly, another set of instructions allow the instruction module to utilise, at self-chosen times, the patterns detected by the perception module to temporarily influence which instructions (external actions, working memory, etc.) will be generated in the following instruction cycles – a process called perceptual advice. Using these instructions, the system can then successfully learn when and how to request further perceptual processing and when and how to make long-term modifications to the perceptual system, thereby achieving active adaptive perception.

Due to fitting the above description, a natural candidate for the instruction module is the self-modifying policy. As discussed in Section 2.5, self-modifying policies such as the Goedel Machine and Incremental Self-improvement learn a policy which includes in its action set instructions for modifying the policy and usually some other instructions which are internal or computational in nature rather than just outputting external actions. If such a policy successfully learns when and how to use self-modification instructions, such a policy learns how to generate changes to itself, then this allows an adaptive learning system which keep learning even when typical reinforcement learning assumptions are not met. Moreover, if such a policy successfully learns when and how to use perceptual modification and perceptual advice instructions, according to their above definition, this will achieve active adaptive perception.

3.3.2 Selective use of representation with multiple policies

The second case study provides a challenging scenario: when partial observability is combined with multiple tasks presented sequentially across the lifetime, with completely different reward functions, knowing how to perform one task may not be beneficial to solving another task. Behaviours that obtain the best performance in some task may well be detrimental in another task. When the task cannot be identified from the observation, simply applying the same policy for a different task might not work. This is easy to see for example in the value-function in Equation 2.2: in partially observable environments the states become limited observations and the reward function varies dependent on the task in an unknown way. With **F** being the current task, reformulating the Q-learning equation accordingly leads to:

$$Q(o_t, A_t, \mathbf{F}) = (1-\alpha) * Q(o_t, A_t) + \alpha(r_t^{\mathbf{F}} + \gamma \operatorname*{arg\,max}_{A_{t+1} \in \mathcal{A}} Q(o_{t+1}, A_{t+1}))\,, \tag{3.1}$$

where the limited observation is denoted by o_t (compare s_t for the full state in Equation 2.2), and the reward is now denoted by $r_t^{\mathbf{F}}$ to illustrate the task-dependency. The equation shows that if the observation does not include any information about the task, then unless the reward function $r^{\mathbf{F}}$ is highly similar, *negative transfer* will happen across tasks and knowledge. For similar reasons *catastrophic forgetting* will occur due to the incremental updates in the Q-learning algorithm: once the value-function has updated to more closely mimic the intermediate tasks, the knowledge of how to perform the original task may be completely lost.

Assuming for a moment that the task at hand is known, then this problem can be solved with two types of solutions. A first solution is to add the task features as parameters to the observation, which has been explored in previous works [151, 42]. A second is to learn multiple policies, which has been explored mainly in three ways:

- Initialise first, specialise later: several methods provide a way to extract an initial policy for a given task and then specialise a separate policy for each task [51, 95, 231, 96, 210].
- Partially defined subpolicies: these methods use differrent policies selectively over the state space [209, 26, 96, 207, 147].
- Learning each task from scratch: this approach, which is not scalable when the number of tasks grows large, is to learn a single policy for each different task, all with different initialisation. Although this may seem simple and sometimes inefficient due to the lack of transfer learning, empirical findings show that this approach usually outperforms the Elastic Weight Consolidation method [95].

As mentioned in the third bullet point, learning a policy for each task can be beneficial compared to state-of-the-art lifelong learning methods, even in Atari domains where observations give a lot of information about the task to solve [95]. Due to the limited scalability of this approach as more and more tasks are given across the lifetime, a natural question that arises is whether performance benefits could be observed by adaptively using a more limited number of polices depending on the task at hand.

Therefore, as illustrated in Figure 3.3, this work considers a fourth alternative, further simply called "learning with multiple policies": rather than defining policies selectively over the state space, they are defined selectively *over a particular subset of tasks*, by using exploration-and-exploitation strategies to learn from the long-term reward intake of policies on tasks which policy has the highest performance on which task and then applying the best policy for a given task most frequently. Due to this selective use of representation, the issue of "Catastrophic forgetting and negative transfer" mentioned above may be avoided or ameliorated. By including an additional layer of adaptivity, which assigns policies to the tasks in which they perform well, the approach is useful to allow policies to specialise to the tasks they are best suited to. Due to defining policies which may diverge towards different regions of the search space, one of the expected benefits of the adaptive approach is to overcome the "Locality of the Search Space" issue mentioned above.

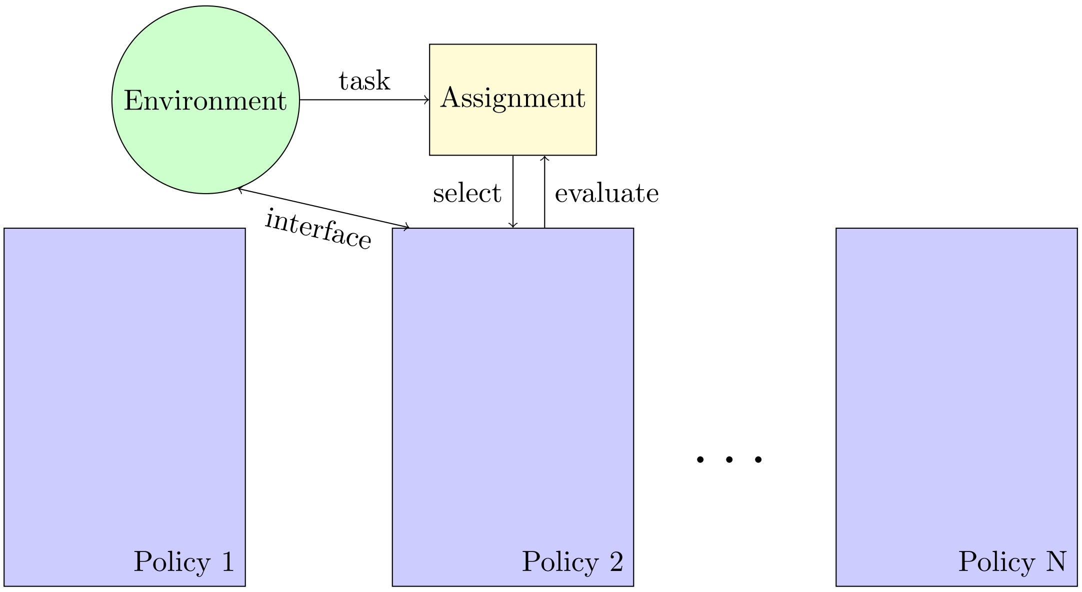


Figure 3.3: *Learning with multiple policies. The following cycle is repeated: first, a policy is chosen to solve the current task; second, the policy interfaces with the environment for a prolonged amount of time (e.g., one episode). Additional adaptivity can be included by allowing a performance evaluation to modify the policy chosen for a given task.*

Furthermore, in the context of the earlier work on capacity, one question remaining is to what extent a reinforcement learning policy can generalise, and particularly, how *many* tasks a single policy would be able to learn. Despite the variety of metrics for function approximation [220, 59, 13], and various proofs about the capacity of neural networks in terms of their ability to represent functions and open dynamic systems [79, 150], capacity metrics are a poor guideline for lifelong reinforcement learning. In reinforcement learning, there are practical difficulties in training procedure, the data that is being presented to learn the patterns, and the objective function which takes into account only a limited time frame or may be biased in a different way.

To assess their capacity in a practical manner, Chapter 5 compares different settings of the number of policies to investigate *task capacity*, how many tasks can be reasonably be learned and remembered by a single policy, in a single lifetime. This notion differs from earlier capacity metrics: first, it assumes a practical setting in which issues related to the learning algorithm, such as catastrophic forgetting and inability to escape local optima, affect the resulting metric score; second, rather than considering how many functions a neural network would be able to learn in isolation, the task capacity metric considers how many tasks can be represented by a single policy at or around the same time. In this sense, the task capacity metric is more similar to the memory capacity in Hopfield networks [59], which captures the number of patterns that can be stored at the same time. Compared to this memory capacity metric, though, there are no architectural constraints and the test is applicable to lifelong reinforcement learning.

Such a task capacity analysis is then used to compare different base learners. By comparing the response of different base learners to the number of policies included, this allows a comparative assessment task capacity as a metric for the goodness of a lifelong reinforcement learner. The use of multiple policies further facilitates additional statistics on the impact of policy diversity and how rapid a base-learner learns and forgets depending on the number of tasks a single policy has been assigned to. This in turn may lead to a better understanding as to why certain objective functions and training methods work and others do not.

3.4 Summary

This chapter presents the main research methodology used in the thesis. The key motivator behind the research methodology is the observation of inductive bias in deep reinforcement learning systems: a limit to effective memory of the causal chain, architectural constraints of the neural networks, the assumption of locality of the search space inherent to methods of the gradient descent type, untargeted exploration, fixed algorithm updates not subject to modification, and catastrophic forgetting and negative transfer.

To investigate how to mitigate these sources of bias, two case studies are explored, a long-term maze where the learner can get stuck forever and a task sequence of randomly presented, unrelated tasks. A novel learning principle called active adaptive perception is proposed, which combines two key features: (i) an active form of adaptive perception by learning how to make modifications to a perception module; and (ii) active perception, by selective request of advice from a perception module. The architecture for active adaptive perception, proposed in this chapter, will be investigated in Chapter 4, as a means to shift the inductive bias for more effective learning on the first case study, but also in Chapter 6, where the issue of catastrophic forgetting is investigated on the second case study. A second learning system that will be explored (see Chapter 5) is learning with multiple policies, a simple method to assign policies to tasks, thereby avoiding locality of search space and negative transfer and catastrophic forgetting. Using this system allows to investigate task capacity, how many tasks a policy can represent, and the potential benefit of adaptively changing the policy to use for a task; and since this method is agnostic to the type of base-learner, it allows investigating the differences between base-learners with regard to these properties.

Chapter 4

Long-term learning with limited knowledge and sparse rewards

In long-term unknown environments, the task presented to the learner will be unknown, meaning that even a single task may be difficult to learn without prior knowledge. As discussed in Chapter 1, one property of general learners is that they abide with the Assumption of Insufficient Knowledge and Resources (AIKR). This principle holds true not only for the learning phase but also for the design phase: if the designer can accurately make many assumptions on the environment, then specialised learning methods will yield better performance than general learners. In the opposite case, avoiding to make assumptions is more important, since assumptions that hold true for some environments will not for others. It is such a case which is the topic of this chapter.

A single unknown task may be presented to the learner for a prolonged amount of time without episodic boundaries and with the possibility of getting no feedback at all, and given the bias of current reinforcement learning systems, this may be misleading. To investigate this scenario, this chapter considers a task which is atypical for reinforcement learners, in the sense that many of the usual assumptions do not hold: terminal states are unknown and there are no time-outs, the environment is partially observable, and the rewards are incurred on a sparse basis with the possibility of not getting feedback at all. Further, unlike some types of Partially Observable Markov Decision Process solvers, such as those in [87, 179, 172], but similar to many other RL methods for partially observable environments, the underlying state space is not known[1]. Due to these properties, it is expected that adaptive learners, which adapt their learning procedures to the environment, will outperform unadaptive learners, which have a bias that is unsuited to this environment.

4.1 Non-episodic partially observable mazes

The maze setting considered in [155, 157], and investigated in this chapter, is an example of an atypical environment. The learner has a lifetime going from $t = 0$ to $t = T$ without any interruptions. The learner initially wanders around without knowing what its goal is. At each time-step, it obtains an observation about whether the four adjacent locations in a Von Neumann neighbourhood[2] are free or blocked and selects an operator from the set of external actions $\mathcal{A}^E = \{\texttt{north}, \texttt{east}, \texttt{south}, \texttt{west}\}$. After taking such an external action, a reward of 1.0 is given only when the learner finds the goal position, otherwise a reward of 0.0 is given. The learners task is to maximise the cumulative reward. Whenever a goal is obtained, the learner is reset to a starting position. However, the environment appears as *non-episodic* to the learner since it has no knowledge of this inherent episodic structure: the goal location is not noted as a terminal state, the memory is not reset after reaching the goal, and there is no artificial time-out to reset the learner when the learner does not reach the goal. Instead, the environment appears to the agent as a single history from $t = 0$ to an unknown $t = T$.

If the above maze setting is extended to larger mazes with similar obstacle density, then this problem is challenging: detracting corridors and rooms, an initially faulty policy, a sparse reward structure, and a non-episodic setting without time-outs, the learning agent may get stuck in a bad region of the maze and experience thousands of steps without any rewards. This is significantly different from other partially observable environments investigated in deep reinforcement learning papers, such as T-maze experiments [12], the 89-state maze [228], Atari experiments [69] and the Invisible Target Capture Task [175]. Those experiments are comparatively easy in the sense that: (a) the agent has knowledge of terminal states and the memory is reset at the start of the episode; (b) to avoid getting stuck without feedback, there is an artificial time-out such that, after a certain number of time steps without reaching the goal, the agent is reset to the initial state; (c) there is no corridor from which it is difficult to escape, instead the space

[1]These methods are a subset of Partially Observable Markov Decision Process solvers which estimate the unknown environment state from the known observations, usually using the "belief state" $p(s_t|h_t)$, the probability of the state given the history of observations and actions. In the below maze, each (x, y)-coordinate's probability could be estimated from the previous Von Neumann neighbourhood observations and the previous actions. Such methods imply that the designer has domain knowledge about which state variables underly the dynamics (x and y in the maze example), even though their exact values are never known.

[2]The Von Neumann neighbourhood around a location (x, y) includes all positions within a Manhattan distance of 1, that is, (x, y), $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$, and $(x, y - 1)$; for observations, only the adjacent locations are considered, because (x, y) is occupied by the agent and therefore redundant.

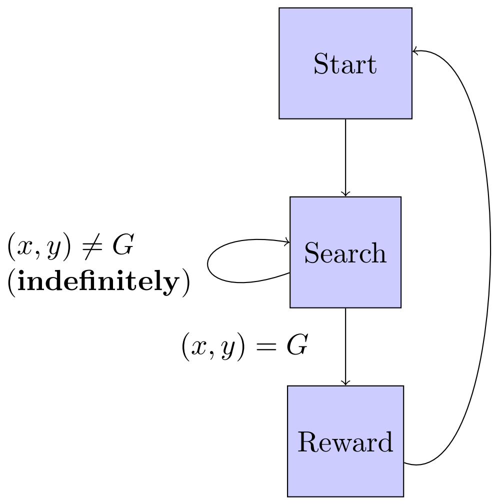


Figure 4.1: *Flow diagram of non-episodic maze with sparse rewards. A random start position initiates the search through the maze for an indefinite period of time, until the goal location is reached. Upon reaching the goal location G, a positive reward is incurred and the agent is reset at a random start, and the search through the maze restarts. A key challenge is that agents may get stuck without rewards for indefinite amount of time, resulting in useless experiennces that do not allow any learning.*

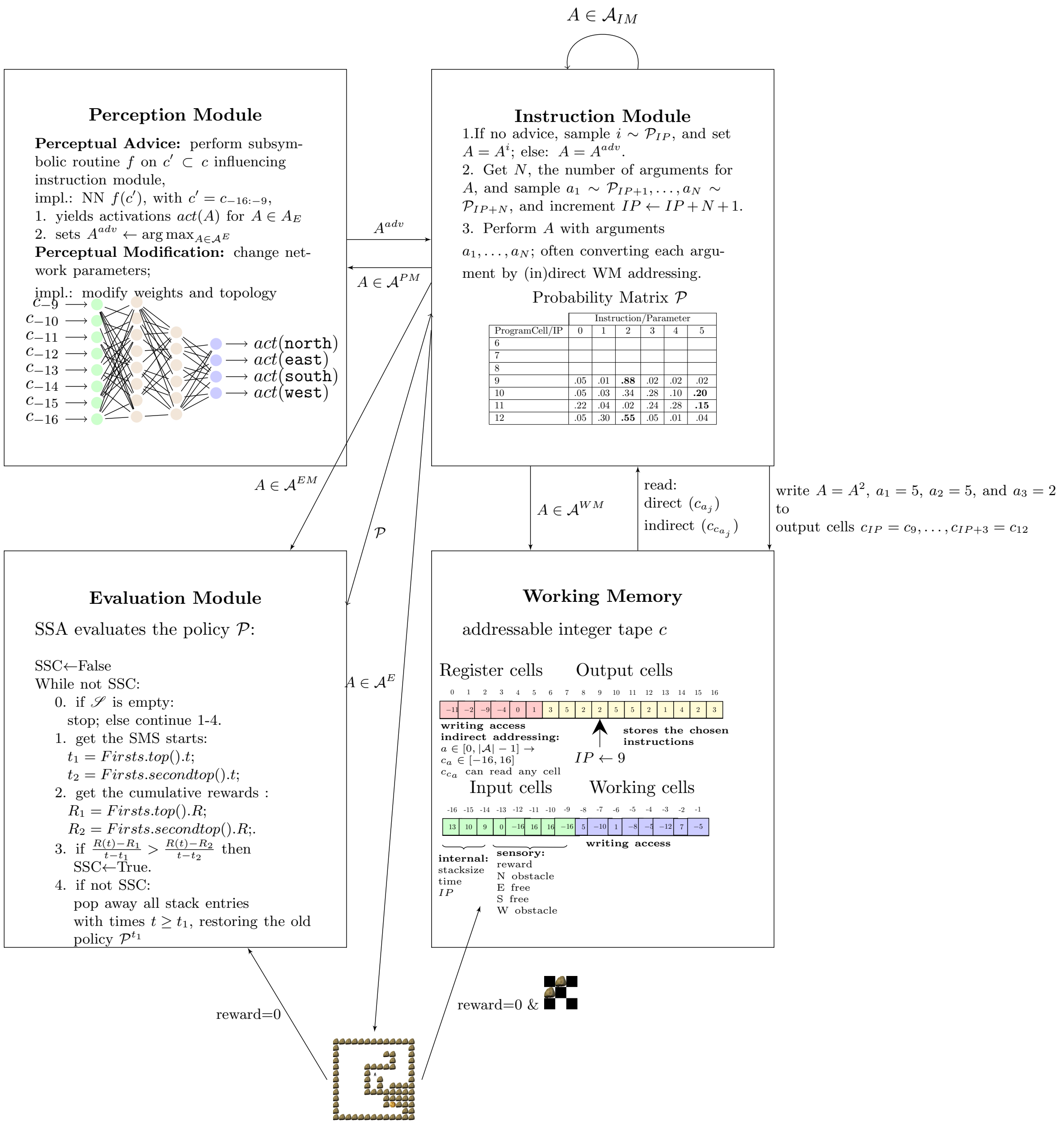
is open or there is a single path; (d) reward structure is more dense, for example, by giving feedback about whether or not the step lead closer to goal. Nevertheless, those experiments have difficulties not addressed in the present maze setting: for example, Atari experiments and the Invisible Target Capture Task have complex dynamics and a large state-space.

4.2 Active Adaptive Perception implementation

This section proposes an exemplary implementation of the generic architecture proposed in Chapter 3, based on Incremental Self-improvement. Figure 4.2 illustrates the meaning of the various modules and their interactions. Two different perception modules will be proposed: the first includes computationally cheap network-modification instructions, illustrate how to learn based on rewards rather than using gradient descent; the second system includes instructions to learn to apply a DRQN learner selectively for particular goals and with particular exploration rates.

4.2.1 Instruction Module implementation: probability matrix

The learner's policy $\mathcal{P}$ consists of a number of m program cells $\mathcal{P}_i$ ($i = ProgramStart$, $\ldots, ProgramStart + m - 1$) each of which represent a discrete modifiable probability distribution over the integers $\{0, \ldots, |\mathcal{A}| - 1\}$ initialised to a uniform distribution but subject to change due to self-modification instructions. Using the instruction pointer


$A \in \mathcal{A}_{IM}$
Perception Module
$c_{-9} \longrightarrow$
$c_{-10} \longrightarrow$
$c_{-11} \longrightarrow$
$c_{-12} \longrightarrow$
$c_{-13} \longrightarrow$
$c_{-14} \longrightarrow$
$c_{-15} \longrightarrow$
$c_{-16} \longrightarrow$
$\longrightarrow act(\texttt{north})$
$\longrightarrow act(\texttt{east})$
$\longrightarrow act(\texttt{south})$
$\longrightarrow act(\texttt{west})$
A^{adv}
$A \in \mathcal{A}^{PM}$
Instruction Module
Probability Matrix $\mathcal{P}$
$A \in \mathcal{A}^{EM}$
$\mathcal{P}$
$A \in \mathcal{A}^{WM}$
read:
direct (c_{a_j})
indirect $(c_{c_{a_j}})$
write $A = A^2$, $a_1 = 5$, $a_2 = 5$, and $a_3 = 2$ to
output cells $c_{IP} = c_9, \dots, c_{IP+3} = c_{12}$
Evaluation Module
$A \in \mathcal{A}^{E}$
Working Memory
addressable integer tape c
Register cells
Output cells
writing access
indirect addressing:
$a \in [0, |\mathcal{A}| - 1] \rightarrow$
$c_a \in [-16, 16]$
c_{c_a} can read any cell
stores the chosen instructions
$IP \leftarrow 9$
Input cells
Working cells
writing access
internal:
stacksize
time
IP
sensory:
reward
N obstacle
E free
S free
W obstacle
reward=0
reward=0 &

Perceptual Advice: perform subsymbolic routine f on $c' \subset c$ influencing instruction module,
impl.: NN $f(c')$, with $c' = c_{-16:-9}$,
1. yields activations $act(A)$ for $A \in A_E$
2. sets $A^{adv} \leftarrow \arg\max_{A \in \mathcal{A}^E}$

Perceptual Modification: change network parameters;
impl.: modify weights and topology

1.If no advice, sample $i \sim \mathcal{P}_{IP}$, and set $A = A^i$; else: $A = A^{adv}$.
2. Get N, the number of arguments for A, and sample $a_1 \sim \mathcal{P}_{IP+1}, \dots, a_N \sim \mathcal{P}_{IP+N}$, and increment $IP \leftarrow IP + N + 1$.
3. Perform A with arguments $a_1, \dots, a_N$; often converting each argument by (in)direct WM addressing.

Probability Matrix $\mathcal{P}$

	Instruction/Parameter					
ProgramCell/IP	0	1	2	3	4	5
6						
7						
8						
9	.05	.01	**.88**	.02	.02	.02
10	.05	.03	.34	.28	.10	**.20**
11	.22	.04	.02	.24	.28	**.15**
12	.05	.30	**.55**	.05	.01	.04

SSA evaluates the policy $\mathcal{P}$:

```
SSC←False
While not SSC:
    0. if $\mathscr{S}$ is empty:
        stop; else continue 1-4.
    1. get the SMS starts:
        $t_1 = Firsts.top().t$;
        $t_2 = Firsts.secondtop().t$;
    2. get the cumulative rewards :
        $R_1 = Firsts.top().R$;
        $R_2 = Firsts.secondtop().R$;.
    3. if $\frac{R(t)-R_1}{t-t_1} > \frac{R(t)-R_2}{t-t_2}$ then
        SSC←True.
    4. if not SSC:
        pop away all stack entries
        with times $t \geq t_1$, restoring the old
        policy $\mathcal{P}^{t_1}$
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
−11	−2	−9	−4	0	1	3	5	2	2	5	5	2	1	4	2	3

-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
13	10	9	0	−16	16	16	−16	5	−10	1	−8	−5	−12	7	−5

Figure 4.2: *Diagram of the active adaptive perception implementation on the maze problem. Based on the current instruction pointer as an internal state, the instruction module generates an instruction and its arguments. The Working memory elements – which form a list of integers in $[-16, 16]$ each with their own address and are organised in four distinct categories (input, working, register and output cells) – are then used to process the arguments for context-sensitive instruction execution. The instruction is then performed, calling the evaluation module to perform SSA, the perception module to modify it or to request an advised action, the working memory to make historical notes, the instruction module to self-modify, or an external action in the environment. When perceptual advice is requested, the perception module's neural network outputs an advised action after taking the input cells' contents, normalised in $[-1, 1]$, as inputs. Input cells are the current reward, the binary observation bits indicating whether north, east, south, and west are free positions or obstacles, and internal variables for disambiguating the state based on the history, namely the time, the stacklength and the instruction pointer. Note that (a) a simplified representation is given because the number of working memory cells and program cells is larger in the experiments; (b) in the SMP-DRQN implementation, the inputs to the perception module is the history of observations instead of all current input cells, and additional perceptual modifications are done on a set of useful experiences.*

IP, a special working memory variable that points to the current program cell, an instruction cycle consists of sampling an integer $j \sim \mathcal{P}_{IP}$ representing an instruction $A^j \in \mathcal{A} = \{A^0, A^1, \ldots, A^{|\mathcal{A}|-1}\}$. After checking how many arguments, N, are required for executing A^j, integer arguments $a_1, \ldots, a_N$ are generated according to the distributions of the following program cells $\mathcal{P}_{IP+1}, \ldots, \mathcal{P}_{IP+N}$. After executing the instruction and its arguments, a new cycle starts with $IP \leftarrow IP + N + 1$. Instructions include various external actions but also internal operations. For self-improvement, the learner uses self-modification instructions `incP` and `decP` which increase and decrease the probability of a chosen program cell $\mathcal{P}_i$ by a chosen amount, respectively. Evaluation is initiated by `endSelfMod` which ends the current modification sequence and starts the evaluation of the latest changes made to the instruction module and the perception module. External actions are application-dependent, such as `north`, `east`, `south` and `west` in maze-problems. The working memory is manipulated using reading, writing and arithmetic operations and instructions that change the IP similarly determine the state of the learner. Perceptual modification and perceptual advice instructions are explained in Section 4.2.4 and 4.2.5, while a complete instruction set is given in Table 4.1.

4.2.2 Evaluation Module implementation: Success Story Algorithm

The function of the evaluation module is to determine if changes to the instruction module and the perception module are beneficial by considering evidence of how self-modifications relate to reward intake.

Implementation: Success Story Algorithm To allow a learner to incrementally improve its performance with minimal assumptions on the environment, an empirical evaluation method called the Success Story Algorithm (SSA) is used which maintains only those incremental modifications that lead to long-term reward acceleration. At time points called checkpoints initiated by the instruction `endSelfMod`, the learner performs an evaluation of the current self-modification sequence (SMS). The learner can adapt to tasks with atypical reward structures because it can determine the frequency of `endSelfMod`, learning how much time is required to reliably evaluate a series of modifications. The evaluation is done using the Success Story Criterion (SSC),

$$\frac{R(t) - R(t_2)}{t - t_2} > \frac{R(t) - R(t_1)}{t - t_1}, \tag{4.1}$$

where $R(t) = \sum_{\tau=0}^{t} r(\tau)$ is the cumulative reward, t is the current time and $t_2 = \mathit{Firsts.top}().t$ and $t_1 = \mathit{Firsts.secondtop}().t$ are the times of the first self-modifications after the most and second-most recent checkpoint, respectively. Thus the SSC asserts whether the reward intake has accelerated since $t_2 > t_1$. If this is true, then modifications made in $[t_2, t]$ will be maintained, otherwise the current modifications are removed and

Instruction	Type	Explanation
`north`	$\mathcal{A}^E$	take one step north
`east`	$\mathcal{A}^E$	take one step east
`south`	$\mathcal{A}^E$	take one step south
`west`	$\mathcal{A}^E$	take one step west
`getOutput`()	$\mathcal{A}^{PM}$	forward inputs $c_{-16:-9}$ through the perception module network, yielding activations $act(A)$ for all $A \in \mathcal{A}^E$. Set $A^{adv} \leftarrow \arg\max_{A \in \mathcal{A}^E} act(c')$; Next cycle the instruction module will execute A^{adv}.
`doQuntil`(a_1,a_2,a_3)	$\mathcal{A}^{PM}$	if $looping = True$ or $t < replayStart$ return; else, set $looping \leftarrow True$, the termination experience $term \leftarrow \mathrm{E}_{a_1}$ as the a_1'th element of the experience set E, the maximal number of looping cycles $until \leftarrow narr(a_2, [1, unroll/2])$, and $\epsilon \leftarrow a_3 * .005$. The next cycles, the DRQN network outputs as the activations $act(A)$ the Q-values $Q(s,a)$ for all $A \in \mathcal{A}^E$ with s denoting the history of observations, and then the ϵ-greedy strategy, with the self-chosen ϵ, selects the next external action. The loop is terminated when the current experience is $term$ or when $until$ time steps have passed.
`weightChange`(a_1,a_2)	$\mathcal{A}^{PM}$	add a copy of the current network to the stack $\mathscr{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes}-1])$,$j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes}-1])$; set $w_{ij} \leftarrow clip(w_{ij} + \mathcal{N}(0,\sigma_w); range_w)$ with $range_w = [-50, 50]$ and $\sigma_w = 5.50$.
`addNode`(a_1,a_2)	$\mathcal{A}^{PM}$	add a copy of the current network to the stack $\mathscr{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes}-1])$,$j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes}-1])$; perform $switch(i,j)$; if $layer(j) > layer(i)+1$ then delete the old connection $(from = i, to = j, w = w_{ij})$, add a new node k in layer $layer(i)+1$ and add connections $(from = i, to = k, w = 1)$ and $(from = k, to = j, w = w_{ij})$.
`addConnection`(a_1,a_2)	$\mathcal{A}^{PM}$	add a copy of the current network to the stack $\mathscr{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes}-1])$,$j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes}-1])$; perform $switch(i,j)$; create a new connection gene $(from, to, w)$ with $w \sim \mathcal{N}(0, \sigma_w)$
`setExperience`(a_1)	$\mathcal{A}^{PM}$	if $t < replayStart$, return; else, add the current value of E_{a_1} to the stack $\mathscr{S}$, the a_1'th element of the experience set, to the stack and replace it with the current experience: $\mathrm{E}_{a_1} \leftarrow (o, A, r, o')$, with o the previous observation, A the previous external action, r the current reward, and o' the current observation.
`incP`(a_1, a_2, a_3)	$\mathcal{A}^{IM}$	push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathscr{S}$. Then, set $\mathcal{P}_{c_{a_1},c_{a_2}} \leftarrow 1 - .01 * c_{c_{a_3}} * (1 - \mathcal{P}_{c_{a_1},c_{a_2}})$, with $c_{a_1} \in \{0,\dots,\lvert\mathcal{A}\rvert - 1\}$; $\mathcal{P}_{c_{a_1},i} \leftarrow .01 * c_{c_{a_3}} * \mathcal{P}$ for all $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$
`decP`(a_1, a_2, a_3)	$\mathcal{A}^{IM}$	push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathscr{S}$. Then, set $\mathcal{P}_{c_{a_1},c_{a_2}} \leftarrow .01 * c_{c_{a_3}} * \mathcal{P}_{c_{a_1},c_{a_2}}$, with $c_{a_1} \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$; $\mathcal{P}_{c_{a_1},i} \leftarrow \mathcal{P}_{c_{a_1},i} * (1 - .01 * c_{c_{a_3}} * \mathcal{P}_{c_{a_1},c_{a_2}})/(1 - \mathcal{P}_{c_{a_1},c_{a_2}})$ for all $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$.
`endSelfMod`()	$\mathcal{A}^{EM}$	evaluate the current self-modification sequence with SSA
`jumpHome`()	$\mathcal{A}^{WM}$	set $IP \leftarrow ProgramStart$
`jumpEq`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	if $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP \leftarrow c_{c_{a_3}}$.
`jumpLower`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	if $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP < c_{c_{a_3}}$.
`add`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} + c_{c_{a_2}}; [MinInt, MaxInt])$
`sub`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} - c_{c_{a_2}}; [MinInt, MaxInt])$
`mult`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} * c_{c_{a_2}}; [MinInt, MaxInt])$
`div`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} // c_{c_{a_2}}; [MinInt, MaxInt])$
`rem`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} \bmod c_{c_{a_2}}; [MinInt, MaxInt])$
`mov`(a_1, a_2)	$\mathcal{A}^{WM}$	$c_{c_{a_2}} \leftarrow c_{c_{a_1}}$
`init`(a_1)	$\mathcal{A}^{WM}$	$c_{a_2} \leftarrow a_1 - ProgramStart - 2$
`inc`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} + 1; [MinInt, MaxInt])$
`dec`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} - 1; [MinInt, MaxInt])$

Table 4.1: *List of instructions used for the instruction set $\mathcal{A}$ in the SMP learners. Instructions are divided in categories based on the module it directly affects: E for environment, PM for perception module, IM for instruction module, EM for evaluation module, and WM for working memory. The SMPs included in the experiments used a different subset of $\mathcal{A}^{PM}$, the instructions relevant for active adaptive perception, and the set $\mathcal{A} \setminus \mathcal{A}^{PM}$ are instructions commonly used in Incremental Self-improvement.* ***Function and operator definitions:*** *c is the working memory tape, often indexed by double/indirect-addressing; $layer(i)$ obtains the layer index of node i; $narr(a, [b,c])$ performs a narrowing conversion from $a \in [0, \lvert\mathcal{A}\rvert - 1]$ to an integer in $[b,c]$; $switch(from, to)$ switches $from$ and to when $from > to$ or aborts the instruction when $from = to$; $\mathcal{N}(\mu,\sigma)$ is the normal/Gaussian distribution; $clip(a;[b,c])$ clips a to an integer in the range $[b,c]$. $a//b$ returns $sign(a) * MaxInt$ if $b = 0$ and integer division otherwise; $a \bmod b$ returns a if $b = 0$ and $a - b * floor(a/b)$ otherwise.* ***Note:*** *some operations yield invalid addresses or numbers according to rules of syntactical correctness (cf. [157]); if these conditions are not met the operation does nothing except for the usual increments to the instruction pointer IP.*

the old instruction module and perception module before t_2 are restored. This is done recursively, until the SSC is met. The complete list of modifications that survived the SSC is maintained in the stack $\mathscr{S}$. The recursive procedure of popping entries that do not yield reward acceleration is illustrated in Algorithm 4.1.

Algorithm 4.1 Success story algorithm for evaluating self-modification sequences (SMSs).
Note that when $sp = 1$, the top entry is compared to the initial entry
$\mathscr{S}[0] = (t = 0, R = 0, \mathcal{X} = \emptyset, first = 0, address = \emptyset)$.

```
procedure popBackUntilSSC
    $sp \leftarrow length(\mathscr{S}) - 1$
    while True do

        if $sp = 0$ then
            break;                                    ▷ no modifications left; SSC satisfied
        else
            $e_1 \leftarrow Firsts.top()$;                           ▷ first entry of the top SMS
            $e_2 \leftarrow Firsts.secondtop()$;                ▷ first entry of the second-top SMS
            if $\frac{R(t)-e_1.R}{t-e_1.t} > \frac{R(t)-e_2.R}{t-e_2.t}$ then
                break;                                 ▷ reward accelerates; SSC satisfied
            else
                popAndRestore;
            end if
        end if
    end while
end procedure
procedure popAndRestore(index $i$)
    while $sp \geq i$ do
        $e = \mathscr{S}[sp]$
        if $e.\mathcal{X}$ is a modification of $\mathcal{P}$ then
            $\mathcal{P}_{e.address} = e.\mathcal{X}$;        ▷ restore probability vector at the correct address
        else
            restore the parameters of the perception module using $e.\mathcal{X}$.
        end if
        $\mathscr{S}.pop()$;                                                  ▷ delete $\mathscr{S}[sp]$
        $sp \leftarrow sp - 1$;
    end while
end procedure
```

4.2.3 Working Memory implementation: addressed integers

The implementation of the working memory is a number of storage cells, each with a unique address in $[Min, Max]$ store integer values in $[-MaxInt, MaxInt]$, initialised randomly and then changed by fixed routines or self-chosen instructions. The functionality of the working memory can be categorised into four types, illustrated in Figure 3.2. Each of the cells have particular integer address to access them, and Section B.1.1

indicates how to obtain the exact addresses used in the experiments. Input cells are special working memory cells that are updated every cycle and which store the current observation as well as other information, i.e., the instruction pointer IP, the time t, the reward r and the length of the stack $\mathscr{S}$. Output cells store the history of recent actions: when $\mathcal{P}_{IP}$ has generated an integer, this number is written to the output cell that is addressed by IP. Working cells provide more long-term memories, only being modified when special instructions in $\mathcal{A}^{WM}$ manipulate them. Register cells have the same function but additionally they are used for a process called double-indexed addressing: because $\mathcal{P}$ generates arguments in the limited range $[0, |\mathcal{A}|-1]$, the values of the register cells in $[0, |\mathcal{A}|-1]$ are used to address the entire range of working memory cells. Working memory contents are used extensively in the execution of the various instructions. For example, the instruction `add(`a_1`,`a_2`,`a_3`)` first reads the contents of the register cell at address a_1 and then fetches the value at the address c_{a_1}, as notated by $c_{c_{a_1}}$; then similarly reads $c_{c_{a_2}}$; finally, adds both $c_{c_{a_1}} + c_{c_{a_2}}$ and stores this sum on the address c_{a_3}. Similar to the above example, many other instructions, including instructions used for self-modification and perceptual modification, also use working memory contents to determine how the instructions are executed.

To illustrate how the processing of the working memory can be used for processing observations to influence external actions, an illustrative example is mentioned for the maze example of Section 2.1:

1. first, the agent records an observation and a reward in its input cells, indicating whether the neighbouring positions are obstacles or free spaces and whether or not it reached a goal;
2. then, it uses working memory operators to manipulate the memory, based on various cells including the working cells;
3. an execution of `jumpEq` or `jumpLower` then sets the instruction pointer IP based on working memory contents;
4. eventually, an external action is executed based on IP.

4.2.4 Perception Module implementation 1: neural network with NEAT representation

The first implementation considers simple instructions to use and modify a feed-forward network, using a representation similar to NEAT [188]. As in NEAT, this implementation represents a neural network using node and connection genes and allows various incremental operators to modify the network; the key difference to NEAT, however, is that no evolution is performed but instead Incremental Self-improvement applies these operators as part of its perception module instructions. Consequently, this implementation highlights the use of active adaptive perception as a framework for constructing and training neural networks within a single lifetime rather than a multitude of independent individuals in a population.

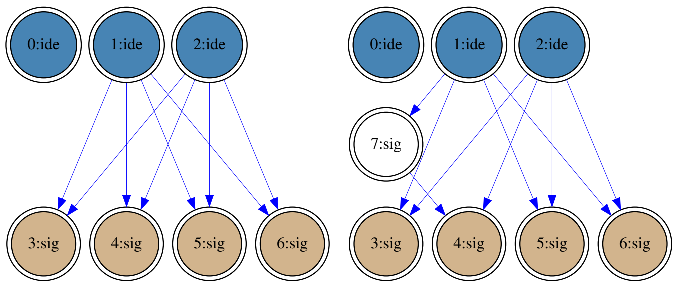
To achieve **perceptual advice** in this implementation, a special instruction `getOutput()` performs forward pass of the perception module's feedforward neural network which takes as input the eight input cells of the working memory and then outputs activations $act(a)$ for each external action. Based on these output activations an advisory action A^{adv} is selected to be executed at the next instruction cycle. A unit u, an elementary node in the network, activates its incoming activation node-input(u,t) at time t according to:

$$u(t) = type_u(\text{node-input}(u,t)), \tag{4.2}$$

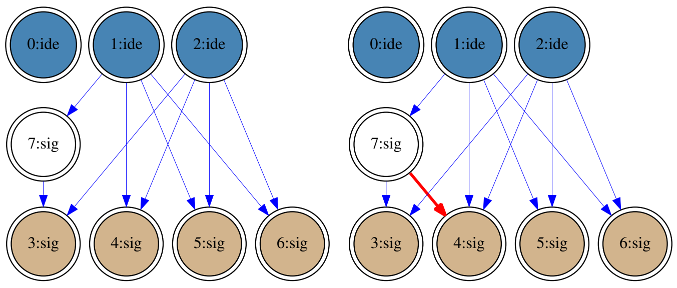
where $type_u$ is the transfer function of u. If unit u is situated at layer ℓ, the node input is defined by:

$$\text{node-input}(u,t) = \sum_{v \in U^{l<\ell}} w_{uv} v(t) \tag{4.3}$$

where $U^{l<\ell}$ is the set of nodes at a layer lower than ℓ. To achieve **perceptual modification**, the learner uses computationally cheap instructions `weightChange`, `addNode` and `addConnection` to modify both topology and weights of a neural network. Each change to the network is recorded on the stack $\mathscr{S}$ such that it can be evaluated later by the evaluation module. This is done with a special representation similar to NEAT, where a neural network consists of two sets of genes. *Connection genes* are tuples of $(from, to, weight)$: *from* and *to* represent the connections input and output unit respectively and *weight* represents the interconnection weight. *Node genes* are tuples of $(type, bias, response, layer)$: *type* is the transfer function used to output at the neuron, *bias* encodes a number to be added to the activation independent of all other incoming activations , *response* is a number that the neuron multiplies with all its incoming weights and *layer* is used to adequately structure the connections. Together, the node genes and the connection genes represent the neural network which is being learned, allowing the use of constructive operators `addNode` and `addConnection`, shown in Figure 4.3(a) and 4.3(b), respectively. Other parameters not changed by the above instructions are fixed, with *type* being sigmoid for non-inputs and identity for inputs, and $bias = 0$ and $response = 4.92$ for all neurons. The weight-range and the response are selected based on the peas-neat implementation, cf. `https://github.com/noio/peas/blob/master/peas/methods/neat.py`, as they are commonly used settings.



(a) `addNode`



(b) `addConnection`

Figure 4.3: *Illustration of the network construction operators. For simplicity, only two input nodes are shown and the bias unit is shown without its connections. Network connections are restricted such that the layers satisfy $from < to$. Blue nodes indicate input units, grey-brown nodes indicate output nodes, and white nodes indicate hidden nodes. Input units use the identity function (denoted by* `ide`*) as a transfer function, while non-input units use the sigmoid function (denoted by* `sig`*). The added connection is emphasised in red bold.*

4.2.5 Perception Module implementation 2: DRQN with modifiable experience set

The second implementation embeds the learning of the Deep Recurrent Q-Networks [69], a recurrent extension to Deep Q-Networks [126], and a modifiable experience set as its perception module. The experience set E is a database of interesting experiences which serve as goals after which network use is halted. This enables selective network usage, learning when to rely on the Q-network, as well as goal-based exploration, learning when to use which exploration rate.

After observing and acting at a time t and $t+1$, DQN fills a buffer $\mathcal{B}$ with experiences (o_t, A_t, r_t, o_{t+1}), which are tuples of observation, chosen action, observed reward, and the following observation. DQN minimises a loss function

$$L(\theta) = \mathbb{E}_{(o,A,r,o')\sim U(\mathcal{B})}[(r_t + \gamma \max_{A'} Q(o', A'; \hat{\theta}) - Q(o, A; \theta))^2] \,, \tag{4.4}$$

where an experience (o, A, r, o') is sampled uniformly from the buffer $\mathcal{B}$ for the next mini-batch update, a process called experience replay; Q is the value-function for Q-learning [226]; γ is the discount used to compute the discounted future cumulative reward $\sum_{i=0}^{\infty} \gamma^i r^i$; θ contains the current parameters whereas $\hat{\theta}$ contains the parameters of the target network which are updated only infrequently, increasing stability. In DRQN, the loss function is the same but the observations are passed through a recurrent network including a Long Short-Term Memory layer [77], such that the history of observations affects the internal state of the learner. The experience replay thus is modified in DRQN to randomly sample a history of *unroll* consecutive experiences rather than a single experience.

Perceptual advice in the second implementation consists of a single instruction `doQuntil`. When the instruction module performs the `doQuntil` instruction the DRQN network takes the observation o as its only input, with no other working memory variables, and outputs the Q-values of the different actions from which it determines the advised action. An ϵ-greedy strategy is used such that with probability $1 - \epsilon$ the chosen action is $A^{adv} = \arg\max_{A_{t+1} \in \mathcal{A}^E} Q(o_t, A_{t+1})$ whereas with probability ϵ a random action is chosen. This is done repeatedly until a self-chosen experience tuple $e_{a_1} = (o, A, r, o')$ is achieved or until a number of self-chosen time-steps are reached without matching e_{a_1}. The arguments of the `doQuntil` instruction determine three important parameters: a_1 determines which experience is taken from the experience set E to end the loop, a_2 determines the number of maximal steps of the loop, and a_3 determines the exploration ϵ. **Perceptual modification** consists of two instructions. The first, `trainReplay`, is the typical experience replay procedure as is performed in DRQN, but with the added flexibility that the instruction module determines the batch size. This instruction is not followed by pushing the network modification onto the stack $\mathscr{S}$ since this instruction already combines updating with an immediate evaluation. The second, `setExperience`, adapts the set of experiences E by replacing the experience at index a_1, E_{a_1}, by the current experience. This instruction then pushes this modification to the stack to allow the evaluation module to perform long-term evaluation of the new E. Note that E is initialised with experiences randomly drawn from the experience buffer just before the replay-start at $t = 50000$.

4.2.6 How the exemplar learns

Instructions `incP` and `decP` modify the probability of a particular entry $\mathcal{P}_{ij}$, and normalisation is then performed ensuring $\sum_k \mathcal{P}_{ik} = 1$. This results in a change of the probability distribution of instructions for a given program cell i. In turn, this changes the system's response to the internal state $IP = i$, a variable changed by the various `jump` instructions and incremented by executing instructions. The probability changes affect not only external but also the internal behaviours due to the choice of instructions and arguments for working memory manipulation, evaluation, perceptual advice, perceptual modification, and self-modification. For self-modification instructions, this leads to a self-referential recursion, meta-meta-...-learning: changes to $\mathcal{P}$ affect how $\mathcal{P}$ will be changed, and so on. In particular, self-modifications may affect the probability of `incP`, `decP` or its arguments for a given internal state IP and given working memory state, implying a context-sensitive learning of (a) self-modification probability; (b) which program cell should be modified; (c) which entry in the program cell's distribution should be incremented or decremented; (d) how large the increment or decrement should be. Because the Success Story Algorithm repeatedly evaluates previous self-modifications and maintains only those self-modifications that result in lifetime reward acceleration, early self-modifications result in better generators of self-modifications later on; Assuming the instructions cover all aspects of learning and behaviour, this means $\mathcal{P}$ improves itself.

4.3 Experimental setup

This section describes the setup for the experiments in this chapter, all of which are based on the examplary non-episodic maze setting explained in Section 4.1.

4.3.1 Time measurement

It is assumed that each action consumes one time-step and computational processes do not consume time, diverging in this respect from [157]. Although the proposed implementation should similarly be applicable to real-time environments, this type of time measurement is used for the following reasons:

- Real-time experiments may overemphasise implementation details rather than assessing whether the learners make efficient use of experience. One algorithm implemented in an efficient language may outperform another simply by virtue of the language.
- Real-time experiments must be carefully designed not to test for trivial behaviours. For example, a simple algorithm which randomly selects an external action, without any learning, may perform better than a more intelligent algorithm, simply because the intelligent algorithm takes more processing time.
- The time measurement based on actions allows direct comparison to traditional reinforcement learning methods which have the same assumption.

4.3.2 Experimental conditions

There are four experimental conditions based on the dimensions *easy* vs *difficult* and *fixed* vs *random*. Easy problems have shorter optima, requiring at least 4-8 steps from start to goal, while difficult problems require at least 11-30 steps from start to goal. Easy problems also have lower ambiguity and fewer free spaces to get lost in. The difficulty is used to test the hypothesis that self-modifying policies are beneficial when environments have higher ceilings of performance. Higher ceilings are defined as a higher potential to increase the reward intake speed compared to a random learner which for every 10^4 time steps has 30-120 rewards for easy and 1-10 rewards for difficult problems. The second dimension, fixed vs random, describes what happens after goal achievement, concretely whether or not the next starting position is fixed or chosen randomly. Ten easy and ten difficult mazes are generated according to Algorithm B.1, found in the Appendix B, on a grid of dimensions 13×13. The resulting mazes have a variety of features: wide open spaces, narrow straight corridors and intersecting corridors which results in central decision points.

For easy problems, each learner was given a lifetime of 5 million time steps because initial experiments suggested learners have converged by then. For difficult problems, optimal path lengths increased approximately four times, and the actual path lengths, and thus the time to learn from rewards, relates exponentially to the optimal path length due to the increase in possible misleading explorations. 80 million time steps are judged to be a reasonable number without excessive computational expense.

4.3.3 Learners

To investigate the impact of various learning properties in the mentioned environments, the following learners are implemented in `python` code:

- **SMP**: the above-mentioned implementation of the generic architecture without perception module is used as the baseline SMP – therefore, its context sensitivity relies only on the instruction pointer and the working memory. This is the same as Incremental Self-improvement (IS) in [157], except the instructions `jump`, effects of which can be achieved using other instructions, and `getP`, an instruction which is rarely included in other experiments.
- **SMP-Fixed**: A perception module is added to the above SMP, to generate an exemplar of active adaptive perception. The perception module is a single feed-forward neural network which outputs external actions whenever the instruction `getOutput` is called, taking as inputs the input cells in the working memory. Thus, the instruction module may generate external actions directly, for example by generating `north`, or indirectly by calling `getOutput`. The network is a fully connected network with two hidden layers of 10 neurons each.
- **SMP-Constructive**: This condition further adds network construction instructions `addNode` and `addConnection` to the SMP-Fixed architecture. Similarly to NEAT, the networks start as a fully connected network without any hidden units.

- **DRQN**: this condition replicates the Deep Recurrent Q-Network with random bootstrapped updates [69]. It was included as an off-policy deep reinforcement learner, using experience replay to more efficiently learn by sampling experiences from an experience buffer and using a target network for improved stability. Two changes are made due to the domain: first, because the observation is small and has no spatial correlations, the convolutional layer is replaced with a dense layer, resulting in a topology of two hidden layers, one dense with 50 RELU-neurons and one LSTM with $tanh$-neurons; second, due to the non-episodic setting, the experience buffer is organised as a single lifetime rather than a multitude of episodes. To implement DRQN, existing code from VizDoom-Keras-RL, cf. `https://github.com/flyyufelix/VizDoom-Keras-RL/blob/master/drqn.py`, was modified to the non-episodic setting and to allow the utilisation of the target-network in experience replay.

- **SMP-DRQN**: to provide a second example of the perception module, this learner utilises the same network as the DRQN condition, but enables the SMP to utilise it selectively as a special loop instruction `doQuntil`, with self-chosen exploration rate and self-chosen termination conditions. The DRQN network is modified using `trainReplay` which performs experience replay with a self-chosen batch size while `setExperience` is used to construct a set of useful experiences for finishing `doQuntil`.

Parameter settings are mentioned in B.1.1, found in Appendix B.

4.4 Experimental demonstration of active adaptive perception

Behavioural assessment Choices of the agents are visually inspected on heat-maps with arrows indicating the most frequently chosen action at each position. In the easy mazes, methods using an LSTM network, namely SMP-DRQN and DRQN are able to memorise the path to the goal, while the other SMPs only learn a basic sense of direction. In the difficult mazes, more differences between the learners emerge:

- SMP has a probabilistic preference for a single default direction which is best leading to the goal;

- SMP-Fixed and SMP-Constructive briefly check detracting corridors before avoiding them, and frequently visit the best corridors. These methods are not completely able to disambiguate their current state, but rather their networks are similar to a Markovian policy in which faulty choices usually do not lead away from goal, and their $\mathcal{P}$-matrix is similar to the SMP, choosing a single direction;

- Early in the lifetime, DRQN memorises the path towards the goal nearly optimally in 4 out of 10 unique mazes, but gets stuck frequently in the other mazes. The detracting corridors and rooms in those mazes are either greater in number or further from goal. Towards the end, two of those unique mazes keep causing problems with getting stuck. These findings are consistent in the sense that the stuck frequency depended reliably on the maze's topology rather than on network initialisation;

- SMP-DRQN similarly has nearly optimal behaviour on those 4 mazes, and only rarely gets stuck in other mazes. The network's output is similar to DRQN but on detracting corridors, where DRQN fails, the method ignores the network and relies on the $\mathcal{P}$-matrix for a global sense of direction, similar to the SMP.

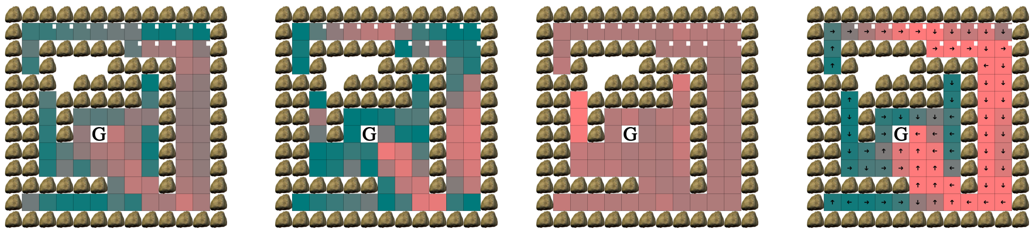
This illustrates the difficulty of traditional SMPs with perception, the difficulty of deep reinforcement learners in atypical environments, and that active adaptive perception may remedy these problems.

A representative example of the final policy is included for one of the most challenging mazes in Figure 4.5, illustrating that methods of active adaptive perception avoid misleading corridors and rooms more often than other methods. Figure 4.4 illustrates behaviours observed for SMP-DRQN during the early to middle stages of the lifetime, showing how SMP-DRQN used its perception module less frequently when it was not reliable. Video material [3] shows the behaviours of DRQN and SMP-DRQN on the mentioned example mazes.

Correctness and perception-correctness The correctness, the proportion of moves that lead the agent closer to the goal, is displayed in Table 4.2. Methods utilising an LSTM network, DRQN and SMP-DRQN, are characterised by relatively high correctness, and their performance was highly correlated with correctness, indicating their performance is dependent on memorising a correct path. For difficult environments SMP-DRQN did not have a positive correlation, suggesting additional strategies beyond path memorisation. This is in line with the observation that the SMP-DRQN has a performance advantage compared to DRQN in the difficult-random condition, where path memorisation is more challenging.

The development of correctness during the early-mid stages of the lifetime illustrate the difference between SMP-DRQN and DRQN. As exemplified in Figure 4.4(b), it can be observed that the perception-correctness, the correctness of the external actions taken due to the perception module's advice, ignoring external actions directly output by the instruction module, varied strongly over the map. The DRQN system, illustrated

[3]https://www.youtube.com/watch?v=xRh-ZXkUJ2Y

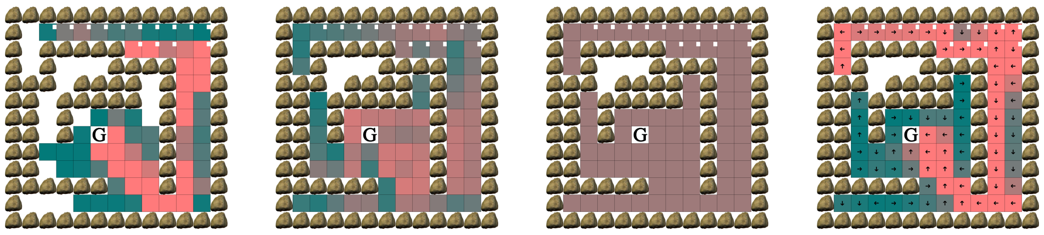


(a) SMP-DRQN network usage (Pr)

(b) SMP-DRQN Perception-correctness (Pr)

(c) SMP-DRQN Exploration rate (Eps)

(d) SMP-DRQN Heatmap (Vis)



(e) SMP-DRQN termination experiences (Vis)

(f) DRQN Correctness (Pr)

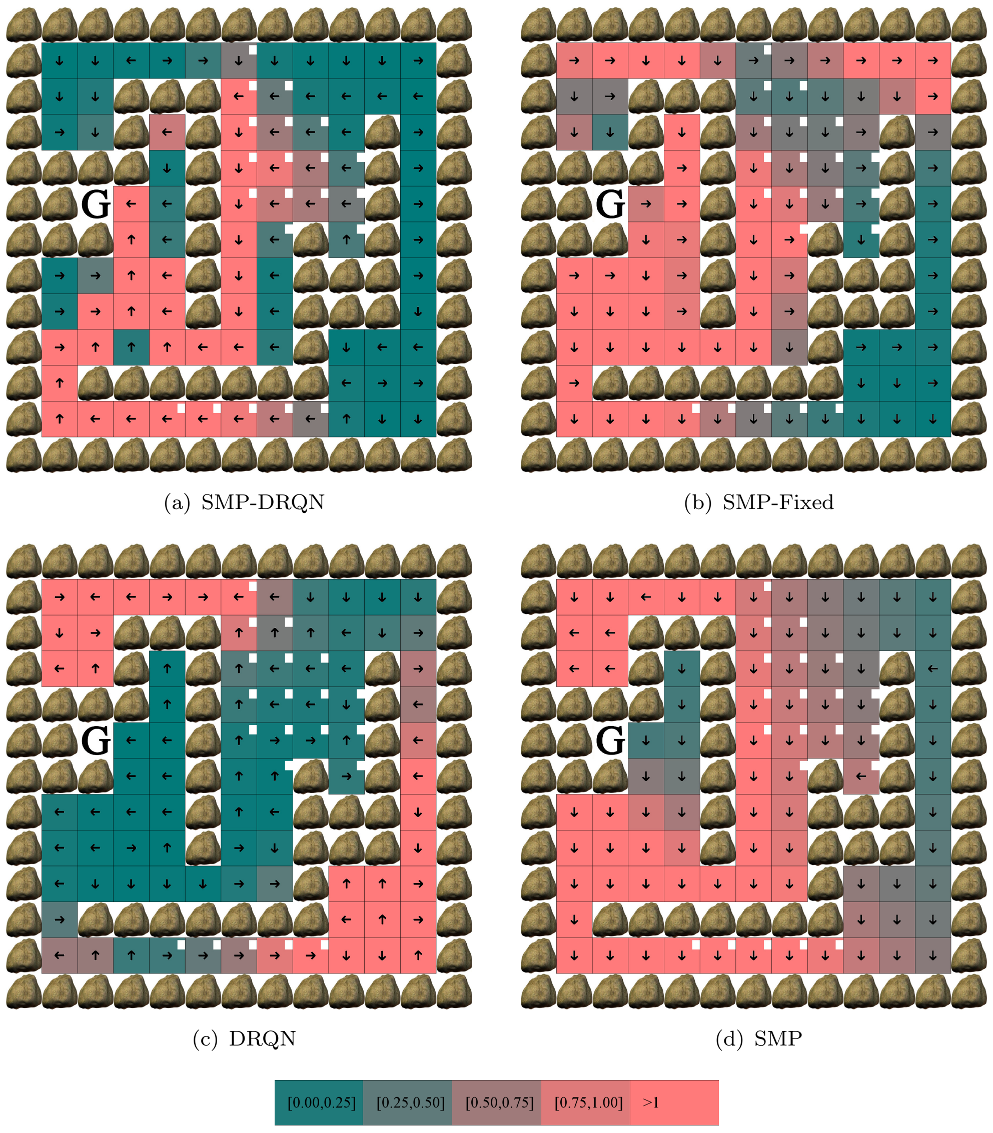
(g) DRQN Exploration rate (Eps)

(h) DRQN Heatmap (Vis)

Pr=[0,.20]	Pr=[0.20,0.40]	Pr=[0.40,0.60]	Pr=[0.60,0.80]	Pr=[0.80,1]
Vis=[0,.25]	Vis=[0.25,0.50)	Vis=[0.50,0.75]	Vis=[0.75,1.0]	Vis>1
Eps=[0,.04]	Eps=[0.04,0.08]	Eps=[0.08,0.12]	Eps=[0.12,0.16]	Eps=[0.16,0.20]

Figure 4.4: *Illustration of the mechanisms behind SMP-DRQN's performance, using the early-mid stage of the lifetime, i.e., in the time interval* $[30 * 10^6, 32 * 10^6]$*, on a maze from the difficult-random condition. The first principle is the* selective network usage*: panel **(a)** shows that the proportion of perception module usage is particularly low in detracting corridors, whilst panel **(e)** shows that this due to how the system matches termination experiences on paths to the goal, halting the network usage before reaching detracting corridors; panels **(b)** and **(f)** illustrate the perception-correctness of SMP-DRQN and the correctness of DRQN is high on paths close to the goal and highly incorrect far from goal and in detracting corridors; together these illustrate that SMP-DRQN uses its perception module selectively on locations with high perception-correctness, ignoring it when it is not reliable. The second principle is the* goal-based exploration*: panels **(c)** and **(g)** illustrate the exploration rate of SMP-DRQN is often higher than DRQN in difficult environments, and especially so on detracting corridors. Together these two principles allow SMP-DRQN to better escape detracting corridors than DRQN, as illustrated in panels **(d)** and **(h)**.*

in Figure 4.4(f), has a low correctness in detracting corridors and rooms, and a high correctness close to the goal, and the same finding is observed for the DRQN network when used as the perception module of SMP-DRQN. The explanation for this finding is that initially in the challenging mazes, the system gets stuck for prolonged time in detracting corridors and rooms, without obtaining any rewards. This leads to erroneous and low Q-values for the visited locations on the map. When later the DRQN system more regularly obtains reward, the detracting corridors and rooms maintain such Q-values for a longer time since they usually do not lead to near-term rewards: far from



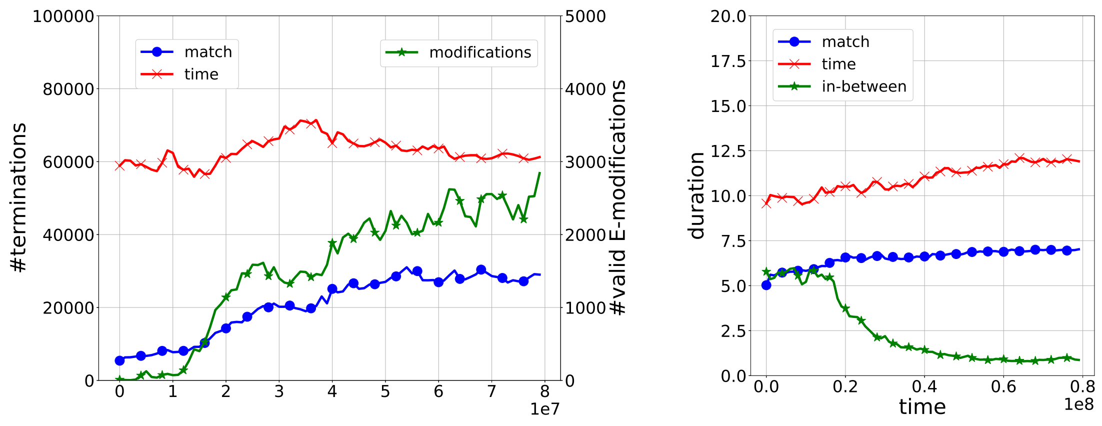
(a) SMP-DRQN (b) SMP-Fixed

(c) DRQN (d) SMP

Figure 4.5: *Heat-map of the final policy on a maze from the difficult-random condition. SMP-Fixed is here taken to represent the first implementation of active adaptive perception since its behaviour is comparable to SMP-Constructive. Though not visible to the agent, the goal location is illustrated by "**G**" while white boxes indicate starting positions. The legend displays the meaning of the colours of the heat-map in terms of visitations per time unit times the number of unique visited locations. $\longrightarrow$: arrows indicate the direction of the most frequently chosen action (north, east, south, or west).*

goal, those corridors and rooms have the lowest Q-values; while close to goal they have lower Q-values than locations which are distant but on path to the goal. By contrast, SMP-DRQN is able to escape detracting rooms and corridors throughout the lifetime due to the mechanisms of selective network usage and goal-based exploration. Because the resulting experiences are added to the experience buffer at each time step, regardless of whether or not the perception module is used, both mechanisms contribute to an intelligent exploration mechanism.

As exemplified in Figure 4.5, the final policies are strongly differing. DRQN still gets



(a) Match (b) Duration

Figure 4.6: *Illustration of the perception module's goal-matching, based on metrics averaged across mazes in the difficult-random condition. Panel (a) illustrates increasing goal-matching: the left y-axis illustrates the number of network usage terminations due to* ***match****, the number of times the system matches the self-chosen termination experience and due to* ***time****, when the looptime of the* `doQuntil` *instruction exceeds the self-chosen until parameter ; the right y-axis illustrates the valid number of* ***modifications*** *to the experience set* E *which contains the termination experiences. Panel (b) illustrates how the duration for* ***match*** *and* ***time*** *increase over time, indicating the learner selects a higher until parameter for the* `doQuntil` *instruction, and how* ***in-between****, the time in between* `doQuntil` *loops, decreases over time as the perception-correctness becomes high across the map.*

stuck in detracting rooms for indefinite periods of time, because it remains inside a detracting room or corridor and does not reach the reward location, filling the experience buffer with useless experiences. After the lifetime of intelligent exploration, SMP-DRQN has now established the optimal policy on key decision points and the path from start to goal, and therefore the SMP-DRQN rarely gets stuck in the rooms.

An additional observation in Figure 4.5 is that when the learner was on the dead-end spaces the DRQN module had low correctness and in some cases more frequent visitation, despite there being only a single action that does not lead to bumping into an obstacle; this occurred either when it was used alone or embedded into SMP-DRQN. This behaviour is due to the combination of two reasons: (i) compared to some other works, for example the T-mazes reported in [12], there is no negative reward incurred for bumping into obstacles or any other incorrect actions, and there is no positive reward for correct actions; and (ii) the incorrect actions do not lead away from the goal at these locations and therefore these actions only delay the reward achievement by one time step, resulting in smaller differences between the different actions' Q-values; this makes dead-end locations more difficult to learn than other locations for which incorrect actions lead to significant delays in reward achievement. The SMP-DRQN was better able to escape such dead-ends by using a high exploration rate and low network-usage at those locations.

Table 4.2: *The correctness metric for the different learners, indicating the proportion of choices made that bring the agent closer to the goal. C_i and C_f are the values of the correctness metric averaged over the various runs during the first and last time slice. The time slice is 1 million time steps for easy and 16 million time steps for difficult. r is the correlation between the lifetime average correctness measure and the lifetime average reward speed.*

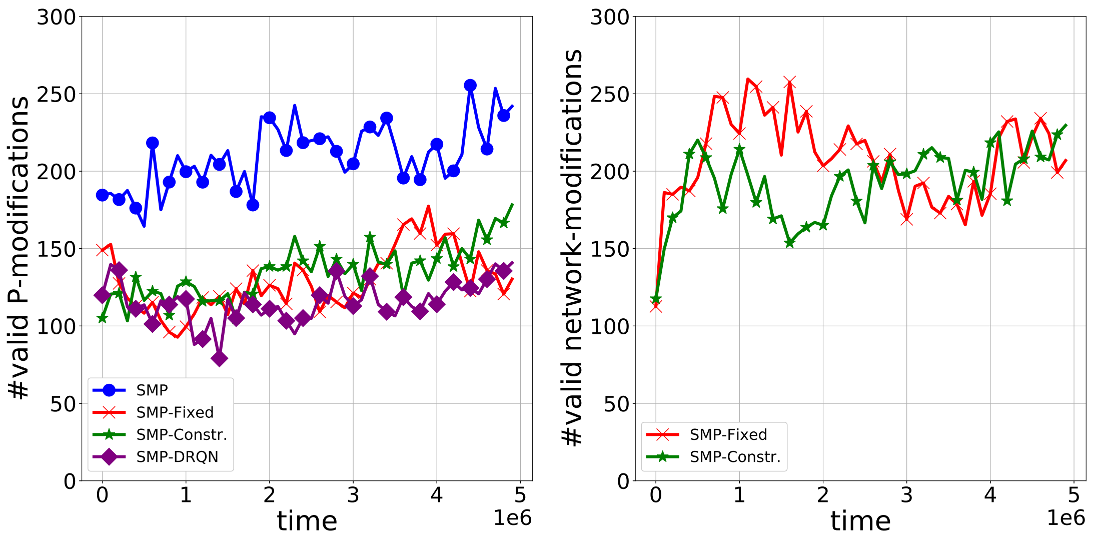
	Environment											
Method	Easy-Fixed			Easy-Random			Difficult-Fixed			Difficult-Random		
	C_i	C_f	r	C_i	C_f	r	C_i	C_f	r	C_i	C_f	r
DRQN	.48	.79	.75	.47	.77	.87	.41	.56	.40	.40	.55	.40
Random	.32	.32	-.69	.33	.33	-.74	.36	.36	.16	.36	.36	.14
SMP	.44	.43	.89	.42	.42	.89	.34	.33	-.67	.33	.32	-.74
SMP-Fixed	.41	.38	.54	.41	.42	.62	.32	.32	.26	.30	.30	.18
SMP-Constructive	.40	.37	.15	.39	.38	.47	.31	.29	-.58	.32	.32	-.06
SMP-DRQN	.61	.72	.93	.58	.71	.95	.38	.55	.06	.37	.62	-.27

Comparatively, SMP-Fixed and SMP-Constructive have a low correctness and, in difficult environments, the random policy, despite its poor performance, has higher correctness than these two methods. Their perception-correctness is high in strategic locations such as paths leading up to the area with the reward or away from a detracting corridor or room, and incorrect decisions usually are not detrimental to performance as can be observed numerically by the absence of positive correlation between reward speed and correctness. This is related to the visual observations that the decisions made usually did not lead further into wrong corridors, which helps to explain the paradox that although the correctness of SMP-Fixed and SMP-Constructive is low their performance is good. For all conditions, the standard deviation of the correctness over mazes is considerably higher for SMP-Fixed and SMP-Constructive than for other methods. This higher variability may indicate that the learning strategy is more dependent on the features of the environment.

Network nodes and connections In the network construction of SMP-Constructive a pattern emerged in which the runs with good performance form a greater number of connections, 2000-4000, and maximise the number of nodes n_{nodes} in the network, specifically 176 for easy problems and 276 for difficult problems (cf. Appendix B for parameter settings). The runs with bad performance would end up with a small number of neurons, 20-90, with the difficult-random condition yielding the lowest cumulative reward and the lowest number of nodes, 20-40. This is supported by the correlation between the number of nodes and the reward speed which is medium to high, 0.60-0.93, over the various conditions. However, there are several exceptionally small networks which resulted in excellent performance. For example, in the difficult-fixed condition, a network of 34 neurons resulted in a lifetime average reward of 0.089 on maze 1 which is much larger than for SMP-Fixed, 0.037, and SMP, 0.013. Since SMP-Fixed is able to perform well with just 20 hidden units, this suggests that constructive modifications are only accepted by the evaluation module to the extent other modification types introduced during that modification sequence are useful.

Network usage The neural network usage, which is the proportion of times the perception module is used to output an external action, developed similarly in SMP-Fixed and SMP-Constructive. It started out small at 20%, but gradually the system started to rely on the network for its instructions, reaching 30% for easy problems and 40% for difficult problems. The discrepancy between easy, 30%, and difficult, 40%, is possibly due to the longer learning time. The heat-maps indicated that the network usage is uniformly spread over the different locations on the map, meaning the learner relied consistently on the perception module. The network usage of SMP-DRQN is higher as advice on several steps are given after a single call of `doQuntil`. In easy environments, SMP-DRQN starts with 5-20% network usage and develops up to 50-70%. In difficult environments, eventually the learner relied on the perception module 90% of the time. Unlike the other methods, the network usage of SMP-DRQN is not evenly spread, especially during the early to middle stages of the lifetime: on areas close to the goal, the network usage is 70-90%, whereas on detracting corridors the network usage is between 20 and 60%. Combined with the fact that the network correctness is much lower in those areas, as illustrated in Fig 4.4(a), this means that SMP-DRQN applies DRQN when it is reliable, such as the paths close to the goal location, but applies a more basic sense of direction where DRQN is not reliable, such as the detracting corridors. This explains why SMP-DRQN performs better in environments where DRQN gets stuck. During the end of the lifetime, the network's correctness in corridors is improved and this resulted in more uniformly high network usage.

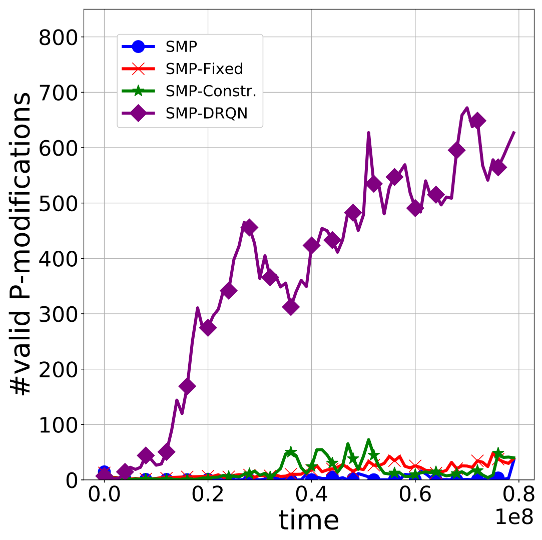
Valid modifications Those modifications maintained at the end of the lifetime, the valid modifications, yield insights into how the agent is learning as they record those changes that accelerated reward intake. These include $\mathcal{P}$-modifications which alter the instruction modules probability matrix and network-modifications which change the network of the perception module. The valid modifications are illustrated in Figure 4.7 and Figure 4.8. In easy problems, the number of valid modifications is spread evenly across time with the different learners making a similar number of valid modifications. The valid modifications are illustrated for the difficult-random environments in Figure 4.8. For all SMPs, a brief initial learning effect is observed, similar to the initial performance gains observed in all learners, since improving on an initial faulty policy is easy. After the initial learning has passed, *learning to learn* is taking place: the learners increasingly learn to generate difficult-to-find modifications that will further accelerate future reward intake. At the end of the lifetime there is a recency effect, a sudden peak in valid modifications as a direct result of halting the lifetime at that point: since recent modifications have only been evaluated a few times, the SSA has not yet removed changes which do not accelerate reward in the long run. Compared to the difficult-random condition, the results for the difficult-fixed condition are more monotonously increasing over time but similarly have a brief initial and final peak. A difference between the learners emerges in the second phase where SMP-Fixed and SMP-Constructive have a much greater number of valid $\mathcal{P}$-modifications, with typical peaks of 25-50 and 50-75, respectively, compared to the traditional SMP with peaks of 5-15. For SMP-DRQN there is a continuously increasing curve, eventually reaching a peak of nearly 700 modifications. This higher amount of $\mathcal{P}$-modifications of the active adaptive perception implementations indicates that most of the useful policy changes involve finding out when and how to modify and utilise the neural network perception module. SMP-Fixed and SMP-Constructive also display a similar pattern on the network-modifications, indicating they have learned how to perform useful modifications to the network weights and topology. Other SMP-DRQN development statistics are mentioned in the following subsection.



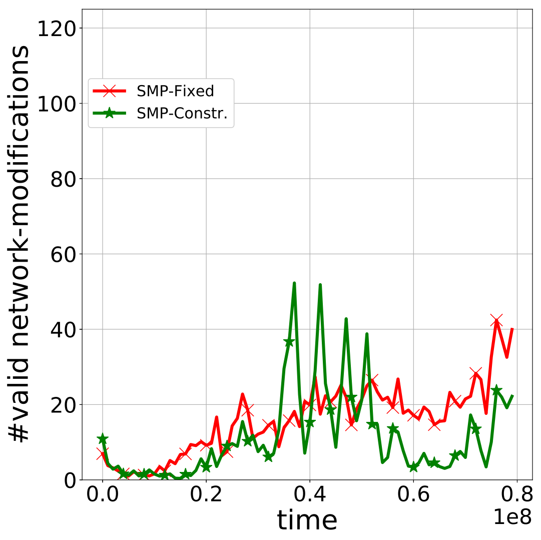
(a) $\mathcal{P}$-modifications (b) network-modifications

Figure 4.7: *Development of the valid modifications in the easy-fixed condition. Valid modifications are those changes that are successful according to the Success Story Criterion, indicating lifetime reward acceleration. Each point in the plot thus represents the number of modifications, introduced in a particular time interval* $[t, t+\delta]$ *with* $t \in [0, T)$*, which remain in use at the end of the lifetime* T*, after repeated SSA evaluations.*

SMP-DRQN development statistics The SMP-DRQN system performs two types of perceptual modifications: `trainReplay` and `setExperience`. `trainReplay` is similar to DRQN's usual experience replay and therefore is not proposed to be the main mechanism behind the performance advantage of SMP-DRQN; this is supported by the lower training frequency exhibited by SMP-DRQN. `setExperience` makes changes to the experience set which are later evaluated by SSA. The `setExperience` and `doQuntil` instruction appear to be key to SMP-DRQN's performance advantage by enabling selective network usage and goal-based exploration. The selective network usage, using



(a) $\mathcal{P}$-modifications



(b) network-modifications

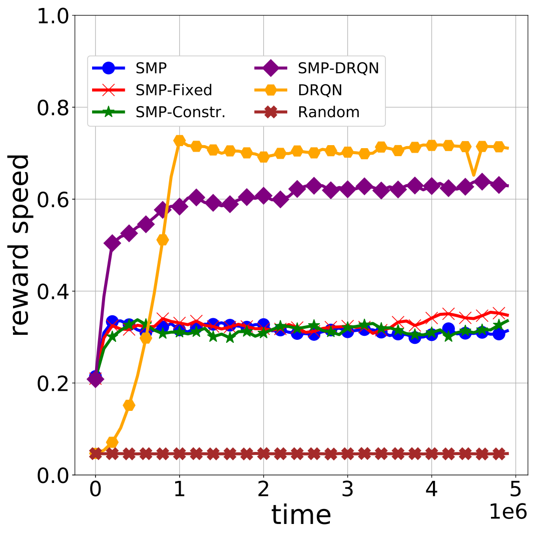
Figure 4.8: *Development of the valid modifications in the difficult-random condition. Valid modifications are those changes that are successful according to the Success Story Criterion, indicating lifetime reward acceleration. Each point in the plot thus represents the number of modifications, introduced in a particular time interval* $[t, t + \delta]$ *with* $t \in [0, T)$*, which remain in use at the end of the lifetime* T*, after repeated SSA evaluations.*

DRQN only where it has reliably memorized the path to the termination experience selected by the instruction module, allows the perception module to be used only when the DRQN is advantageous. In areas where DRQN performs poorly the instruction module can directly output external actions. This allows, for example, escaping rooms where the DRQN is stuck. A second factor is goal-based exploration. This allows the instruction module to determine which exploration rates should be chosen together with which termination experiences, meaning that high exploration rates can be set in areas where the learner does not recognise where it is or what is the best action. In the maze tasks, these two factors allow the system to escape detracting corridors and rooms, finding more rewards. Due to reaching the reward location more often initially, these learners can also accumulate more useful experiences compared to learners which get stuck.

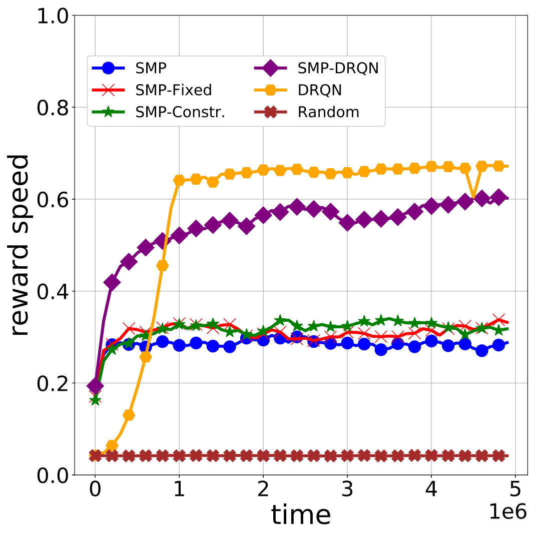
The selective network usage is enabled by the instruction module selecting the `doQuntil` instruction and its two key parameters: the *term* experience, an experience taken from the experience set E, and the *until* parameter, a time limit to network usage. The `doQuntil` instruction then repeatedly requests external actions from the perception module until the current experience matches *term* or until the loop time exceeds *until* time steps. The results of this matching process are illustrated in Figure 4.4(e), where it can be seen that the successfully reached *term* experiences include strategic locations on the path from start to the goal, avoiding usage in detracting corridors. When the *term* experiences are not matched, the network is not used for prolonged amounts of time in detracting corridors and rooms due to the time limit of *until* time steps. Figure 4.6(a) further illustrates that the system is able to better match the self-chosen termination experiences over time. This is not only due to the `setExperience` instruction modifying the experience set E and the increasing network-correctness due to the experience replay, but also, as illustrated in Figure 4.6(b), due to the $\mathcal{P}$-modifications, which increase the *until* parameter to allow itself more time to reach the more difficult goals. Figure 4.6(b) also illustrates why towards the end of the lifetime, the network usage is uniformly high: as the network's correctness increases, the system learns it can boost the reward speed by increasing the *until* parameter and the frequency of the `doQuntil` instruction.

The goal-based exploration is enabled by the instruction module's choice of the exploration rate, as the third parameter of the `doQuntil` instruction, together with the self-chosen *term* experience which serves as a goal. Illustrative of this principle, the exploration rate is dependent on the difficulty of the environment and the chosen termination experiences: in easy environments rates are lower, with some experiences having an exploration rate between 0.02 and 0.05, most around 0.05-0.11, and the highest average exploration rate is $\epsilon = 0.12$, whereas in difficult environments, most experiences are associated with an exploration rate between 0.09 and 0.12, some are between 0.02 and 0.08, and others between 0.13 and 0.16. This suggests that the system learns which termination experiences are more difficult to achieve and therefore require more exploration. This finding is supported by exploration maps such as those in Figure 4.4(c), where it can be observed that detracting corridors have relatively high exploration rates compared to DRQN.

Average performance The development plots in Figures 4.9 and 4.10 display the development of reward speed, the average reward per time step, divided by the optimal reward per time step. On the easy mazes, SMP-DRQN and DRQN obtain the highest reward speeds. DRQN obtains a final reward speed close to 0.70 while SMP-DRQN is just above 0.60. Other SMP conditions are just above 0.30. In the difficult problems SMP-DRQN and DRQN are by far the top performers on the average reward speed. DRQN obtains a final reward speed around 0.4 while SMP-DRQN obtains reward speeds of 0.4 and 0.5 in the fixed and random condition, respectively. Compared to the development in easy problems, more differences emerged between the different SMPs. SMP-Fixed and SMP-Constructive are continuously improving across the lifetime while SMP only initially found good policy improvements. In the fixed condition, this leads to a final reward speed of 0.1 for SMP-Fixed and 0.08 for SMP-Constructive, while SMP has a speed of 0.025. The random starting position gives a similar performance for SMP, 0.02 across the lifetime, .08 for SMP-Fixed and 0.07 for SMP-Constructive. In difficult problems, it can also be observed that while SMP-Constructive initially learns more quickly, its learning rate slows down compared to SMP-Fixed after around $5 * 10^6$ steps.

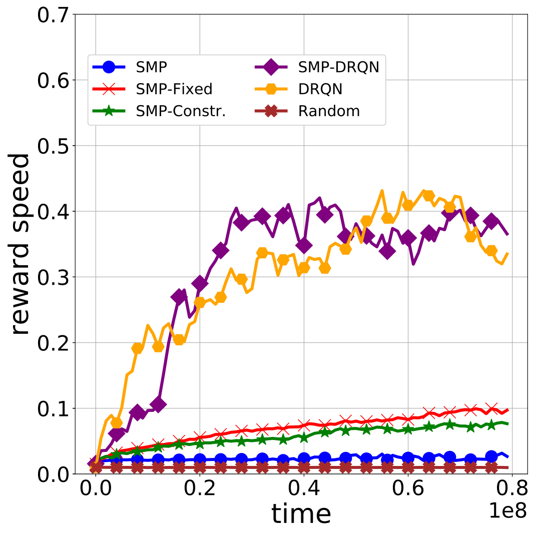


(a) Fixed

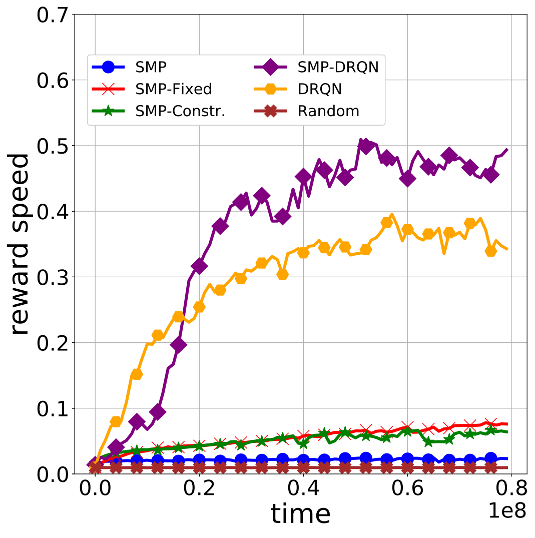


(b) Random

Figure 4.9: *Development plots of the reward speed for the easy-fixed and the easy-random condition, over the lifetime of 5 million time steps. For each plot reward speed, the average reward per time step, is averaged over 20 runs, 2 repetitions for each of the 10 mazes, and normalised in* $[0, 1]$ *such that the optimal speed gives performance of* 1.0.



(a) Fixed



(b) Random

Figure 4.10: *Development plots of the reward speed for the difficult-fixed and difficult-random condition, over the lifetime of 80 million time steps. For each plot reward speed, the average reward per time step, is averaged over 20 runs, 2 repetitions for each of the 10 mazes, and normalised in* $[0, 1]$ *such that the optimal speed gives performance of* 1.0.

As illustrated in Table 4.3, DRQN obtained the best lifetime average in the easy environments, 0.615 (fixed) and 0.572 (random), but SMP-DRQN obtained the best lifetime average in difficult environments, 0.310 (fixed) and 0.361 (random). Table 4.3 further shows pair-wise F-tests conducted on the lifetime average reward speed to analyse whether or not between-condition variability is significantly higher than within-condition variability. For the easy problems, no significant effects are found except for the SMP-DRQN and DRQN learners which significantly outperform all other learners. In the difficult problems, the performance of both SMP-Fixed and SMP-Constructive leads to significant effects when compared to SMP. This indicates that rather than maze variability, the principle of active adaptive perception explains why SMP-Fixed and SMP-Constructive outperform SMP. In turn, the difference between SMP-DRQN and DRQN is not significant while pair-wise differences of these learners to SMP-Fixed and SMP-Constructive are significant.

Table 4.3: *Variance analysis on the effect of learning condition on lifetime averaged normalised reward speed (mean ± standard-deviation). $<$ and $>$ are used to indicate whether the method's performance is higher or lower than its comparison, while d is the effect size and p denotes the significance value of the pair-wise F-test.*

	Method	**Performance**	**Comparison**				
			DRQN	SMP-Constr.	SMP-Fixed	SMP	Random
Easy-Fixed	SMP-DRQN	0.592 ± 0.132	$< p = 0.644$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.615 ± 0.069	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.312 ± 0.120	/	/	$< p = 0.807$	$< p = 0.982$	$> p < 0.001$
	SMP-Fixed	0.325 ± 0.111	/	/	/	$> p = 0.813$	$> p < 0.001$
	SMP	0.313 ± 0.106	/	/	/	/	$> p < 0.001$
	Random	0.046 ± 0.019	/	/	/	/	/
Easy-Random	SMP-DRQN	0.542 ± 0.164	$< p = 0.649$	$> p = 0.003$	$> p = 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.572 ± 0.113	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.314 ± 0.142	/	/	$> p = 0.923$	$> p = 0.611$	$> p < 0.001$
	SMP-Fixed	0.308 ± 0.124	/	/	/	$> p = 0.657$	$> p < 0.001$
	SMP	0.284 ± 0.118	/	/	/	/	$> p < 0.001$
	Random	0.042 ± 0.019	/	/	/	/	/
Difficult-Fixed	SMP-DRQN	0.310 ± 0.109	$> p = 0.909$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.304 ± 0.135	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.056 ± 0.027	/	/	$< p = 0.425$	$> p = 0.001$	$> p < 0.001$
	SMP-Fixed	0.069 ± 0.041	/	/	/	$> p = 0.002$	$> p < 0.001$
	SMP	0.023 ± 0.013	/	/	/	/	$> p < 0.001$
	Random	0.010 ± 0.004	/	/	/	/	/
Difficult-Random	SMP-DRQN	0.361 ± 0.075	$> p = 0.294$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.295 ± 0.176	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	$0.050 + 0.030$	/	/	$> p = 0.765$	$> p = 0.011$	$> p < 0.001$
	SMP-Fixed	0.054 ± 0.029	/	/	/	$> p = 0.003$	$> p < 0.001$
	SMP	0.021 ± 0.013	/	/	/	/	$> p = 0.008$
	Random	0.010 ± 0.004	/	/	/	/	/

Other performance metrics The average reward speed, even when normalised, does not necessarily imply superiority, because an excellent relative performance in the most difficult environment will not contribute as much as an excellent absolute performance in a less challenging environment. To resolve this issue, additional metrics illustrate this comparison in Table 4.4. In the easy mazes, it is clear that DRQN performs the best on all metrics, followed closely by the SMP-DRQN; a more extended lifetime could potentially overcome this given the trend in both development plots. In the difficult mazes, SMP-DRQN has the best average rank, scoring among the top performers consistently, and is followed by DRQN and SMP-Fixed which have the same average rank. The performance ratio, the ratio of the method's average performance to the average performance of the best ranked method, illustrates that SMP-DRQN has the best relative performance, followed by DRQN. Finally, the stuck frequency measures how prone the learner is to get stuck without obtaining rewards; on this metric, the DRQN learner clearly performs worst. To illustrate the statistical significance of the stuck frequencies, pair-wise F-tests comparing the SMPs to the DRQN learner yielded $p = 0.092$ for SMP, $p = 0.036$ for SMP-Fixed, $p = 0.065$ for SMP-Constructive and $p = 0.094$ for SMP-DRQN in the difficult-fixed condition, and $p = 0.063$ for SMP, $p = 0.035$ for SMP-Fixed, and $p = 0.034$ for SMP-Constructive and $p = 0.033$ for SMP-DRQN in the difficult-random condition. This means that based on a threshold for significance $\alpha = .05$, all learners with active adaptive perception have significantly lower stuck frequency in the difficult-random condition, supporting observations that they avoid detracting corridors and rooms more easily.

Table 4.4: *Additional performance metrics (mean ± standard-deviation), for the different conditions (a) easy, and (b) difficult.* ***rank*** *indicates the rank, ranging between 1.0, always best, and 6.0, always worst.* ***ratio*** *indicates the ratio of performance to the performance of the best of both, yielding 1 if it is the best, otherwise a number in* $[0, 1)$*.* ***stuck*** *is the proportion of consequent samples in which the cumulative reward did not increase, with a sampling rate of once every* 10000 *time steps.*

(a) Easy

	Fixed			Random		
	rank	ratio	stuck	rank	ratio	stuck
DRQN	1.2 ± 0.4	$.97 \pm .08$	$0.00 \pm .00$	1.4 ± 0.5	$.97 \pm .06$	$0.00 \pm .00$
Random	6.0 ± 0.0	$.07 \pm .03$	$0.00 \pm .00$	6.0 ± 0.0	$.07 \pm .03$	$0.00 \pm .00$
SMP	3.9 ± 0.8	$.49 \pm .16$	$0.00 \pm .00$	4.3 ± 0.8	$.46 \pm .16$	$0.00 \pm .00$
SMP-Fixed	3.9 ± 0.8	$.50 \pm .16$	$0.00 \pm .00$	3.7 ± 0.8	$.51 \pm .18$	$0.00 \pm .00$
SMP-Constructive	4.2 ± 0.8	$.48 \pm .16$	$0.00 \pm .00$	3.8 ± 0.8	$.52 \pm .21$	$0.00 \pm .00$
SMP-DRQN	$1.7 + 0.4$	$.91 \pm .12$	$0.00 \pm .00$	1.7 ± 0.7	$.90 \pm .14$	$0.00 \pm .00$

(b) Difficult

	Fixed			Random		
	rank	ratio	stuck	rank	ratio	stuck
DRQN	1.9 ± 1.4	$.63 \pm .43$	$.20 \pm .25$	2.3 ± 1.9	$.74 \pm .39$	$.24 \pm .29$
Random	5.8 ± 0.5	$.04 \pm .07$	$.01 \pm .01$	5.7 ± 0.4	$.05 \pm .03$	$.01 \pm .01$
SMP	4.8 ± 0.6	$.08 \pm .09$	$.04 \pm .12$	4.9 ± 0.5	$.06 \pm .03$	$.05 \pm .10$
SMP-Fixed	3.2 ± 0.5	$.22 \pm .16$	$.02 \pm .03$	3.2 ± 0.5	$.14 \pm .08$	$.03 \pm .06$
SMP-Constructive	3.6 ± 0.9	$.20 \pm .20$	$.04 \pm .07$	3.3 ± 0.7	$.13 \pm .08$	$.04 \pm .06$
SMP-DRQN	1.7 ± 0.9	$.83 \pm .23$	$.05 \pm .05$	1.5 ± 0.5	$.93 \pm .09$	$.03 \pm .02$

Resource consumption Experiments on the difficult problems lasted 20-60 hours for most SMP runs, 20-25 days for SMP-DRQN runs and 40-50 days for DRQN runs, using a single Intel Xeon E5-2670 CPU (2.60GHz) on the IRIDIS4 supercomputer [215]. The memory requirements of SMPs can grow strongly over time. As an illustration, the average number of valid modifications for SMP-Fixed and SMP-Constructive is higher than for SMP, 10507 elementary modifications for SMP, 16846 for SMP-Fixed and 16362 for SMP-Constructive in the easy-fixed condition, and 215 elementary modifications for SMP, 2556 for SMP-Fixed and 2225 for SMP-Constructive in the difficult-random condition. For SMP-Fixed and SMP-Constr, half to two-thirds of modifications are network-modifications, which use more memory, illustrated by the stack's memory consumption of 1MB-1.8 GB as opposed to 1-6MB for SMP. SMP-DRQN has even more valid modifications, mainly because of the many changes to the experience set. To examplify, the number of valid modifications to the experience set averaged 121683 in the difficult-random condition. However, its memory consumption is not so high due to the simplicity of the experience set modifications, namely 80-350 MB in the difficult conditions and 60-90MB in the easy conditions.

4.5 Discussion

4.5.1 Strengths

As illustrated by experiments, examplars of the SMP-DRQN type exhibit a greater flexibility and higher performance compared to DRQN in the difficult mazes. This is because of three key factors. First, this learning exemplar makes use of repeated long-term evaluation, repeatedly calling the SSA evaluation module to evaluate self-modifications in terms of their contribution towards the lifetime reward intake. This is especially useful in sparse rewards environments because behaviours that lead to getting stuck without rewards for extended periods of time will eventually be removed. Second, the exploration rate is determined dependent on the difficulty of the local environment to allow exploitation where the pattern is already learned and exploration where the pattern is still to be discovered by the learner. Third, the selective network usage allowed to apply selectively the DRQN only where it is reliable, and this allowed it to avoid the faulty policy locally in detracting rooms and corridors, avoiding to get stuck.

The SMP-Fixed and SMP-constructive have lower average performance but it is surprising to see that given the limited instructions for learning they are still able to display unique behaviours; despite utilising a simple feedforward network which is usually not suitable for partially observable environments, and utilising only simplistic instructions, due to active adaptive perception and long-term evaluation they are able to avoid detracting corridors and rooms; this is in contrast to the traditional DRQN system which bases its decisions and training on expensive LSTM forward and backward propagation procedures.

One of the conjectures of the thesis is that active adaptive perception is preserved even when the implementation is changed significantly; due to the architecture's genericity, different implementations may be used for each of its four components. For example, the evaluation module need not be the Success Story Algorithm. The perception module may consist of not one but several sub-symbolic components such as neural networks, support vector machines or clustering methods, and perceptual advice does not necessarily output external actions, but may be any operation which temporarily influences instruction generation. For example, it may make temporary changes in the probabilities of neighbouring cells or change the contents of internal variables to generate instructions based on a classification of the agent's state. The instruction module may generate its programmatic instructions using a representation different than a probability matrix. The working memory could be implemented differently to use real numbers instead, allowing to solve environments with continuous state and action spaces. Similarly, the interactions between the components may be directed by a different set of instructions. Additional components suitable for cognitive architectures may be added for further gains in complex tasks. For example, a long-term memory could potentially enable more long-term planning and improved behaviour on coarse time scales. To enable large-scale experimentation whilst maintaining the incremental network parameter updates with long-term evaluations, as in the NEAT-network implementation, future work could modify a functional, abstract representation of a network rather than a network itself, similar to HyperNEAT [187]. Moreover, its random increments to the network parameters could be improved: a straightforward extension to the NEAT implementation could be to, in addition to learning which parameters are in need of update, also learn how to increment or decrement the parameters by including the increment as an additional argument to the instruction.

The proposed active adaptive perception implementations are expected to be useful for real-time environments. For example, the proposed `trainReplay` instruction would be interesting in real-time scenarios, because the Incremental Self-improvement learner can decide when to use the instruction, thereby allowing a balance between real-time constraints and efficient use of experience. This point is not speculative, because earlier implementations of Incremental Self-improvement with SSA [157, 166] have demonstrated the ability to optimise the real-time performance.

4.5.2 Limitations

Self-modifying systems are inherently more opaque and therefore more difficult to understand than traditional reinforcement learning systems, because they are not provided with a fixed learning routine to follow, and therefore their learning behaviour may be different for each different run. Such systems therefore require additional caution in realistic environments, and analysing such systems in complex dynamic environments is challenging.

Currently, two aspects of the current implementations scale poorly towards large problems with continuous variables: the working memory so far is limited to integers and modifies a single variable at a time, and the goal-matching procedure works on a limited set of experiences which may be more difficult to match as the observation or action space grows.

Due to modifying the perception module one parameter at a time, the NEAT implementation is not suitable for large-scale experiments. The feedforward structure did not solve partial observability, despite including historical variables, and instead additional working cells as inputs or a recurrent structure should be considered. In addition, despite often introducing more complexity, the performance of the constructive network is comparable to a network with fixed topology. One reason may simply be due to the nature of constructive neural networks which tend to learn fast initially but resulted in a similar even sometimes lower final performance due to overfitting on the initially small network [55, 86]. In addition, the observed relation between reward intake and addition of nodes and connections suggests that SSA is not noticing small negative effects of constructive changes that go together with large positive effects of weight and instruction probability changes, due to the evaluation of modification sequences rather than individual modifications.

Although the study experiments with different perception modules, the study does not address the full scope of active adaptive perception. The evidence brought forward in this chapter does not necessarily indicate there may be benefits for other aspects of active perception, such as the use of predictive models to predict future sensor activations and repositioning the sensors.

4.6 Summary

In long-term unknown environments, the task presented to the learner will be unknown, meaning that even a single task may be difficult to learn without prior knowledge. This chapter investigates a single unknown task presented to the learner for a prolonged amount of time without episodic boundaries and with the possibility of getting no feedback at all. Given the bias of current reinforcement learning systems, this may be misleading and therefore it is investigated how adaptive learning may be used to overcome this bias. To this end, the chapter investigates active adaptive perception, the ability to modify and utilise perception modules in completely self-chosen ways. The chapter demonstrates two exemplar systems with active adaptive perception, according to the generic architecture presented in Section 3.3.1, which are compared to other methods on non-episodic partially observable mazes with sparse reward structures. The first exemplar learns to modify and use a feedforward network with a simple instruction set based on long-term reward intake of the self-modifying policy, instead of traditionally training the network on an explicit loss function. This has lead to simple strategies to avoid detracting corridors and rooms, comparing favourably over a traditional self-modifying policy. A second exemplar system is more computationally expensive, using a recurrent network and experience replay. This system uses instructions to determine when and how to apply and update a DRQN network. It learns to selectively apply the DRQN where it is reliable and to select the exploration factor depending on its current goal. This is shown to be beneficial compared to DRQN on the most difficult problems included, where DRQN gets stuck in detracting corridors and rooms. The architecture also constitutes a novel framework for training and constructing neural networks by learning to use elementary user-defined instructions based on long-term reward intake.

Chapter 5

Lifelong learning with multiple policies

In long-term unknown environments, there may be multiple unknown tasks presented in sequence. In such lifelong learning environments, it is essential to selectively transfer knowledge only to tasks which are related, thereby stimulating positive transfer and avoiding negative transfer. Similarly, it is essential to avoid catastrophic forgetting, where knowledge of a task learned earlier may be completely erased.

To improve performance of reinforcement learners and allow the development of analysis tools, this chapter investigates the use of multiple policies, each assigned to a subset of the tasks. Beyond providing a means to provide improved performance in lifelong learning environments, this "learning with multiple policies" approach will be used to assess two questions:

- Task capacity: How many tasks can a single policy represent?
- Adaptive task assignment: Is it better to (a) train a policy on a static, predetermined set of tasks, which limits the number of tasks to train on and thereby increases the number of training samples per task; or (b) to explore each policy on all tasks, which allows searching for each policy the best set of tasks but which "wastes" some training samples on exploratory tasks on which the policy eventually does not specialise?

5.1 Learning many tasks with a limited number of policies

This section explains the strategy used to learn many tasks with a limited number of policies, and how it provides a metric for task capacity, the number of tasks a policy can represent from a set of tasks corresponding to different coordinates in a space of tasks forming the application domain of interest.

5.1.1 Algorithmic overview

The user can select a number of N_{pol} policies. The top-level learner $\mathcal{L}$ receives an identifier $i \in \mathbb{N}$ corresponding to the current task $\mathbf{F}_i$ from the environment $\mathcal{E}$. Then at regular intervals, called policy intervals, it selects the current policy according to a mapping $M : \mathbb{N} \to \{1, \dots, N_{pol}\}$. The policy then behaves as a usual reinforcement learning policy. Tasks are assumed to be episodic and mutually independent in the sense that the performance on one task does not affect the states or performance on any other task.

5.1.2 Unadaptive assignment of policies to tasks

One special case of the policy-selection map M investigated in this study is a fixed deterministic mapping of a set of tasks to a single policy. For $N_{pol} = 1$ this means a single policy is used throughout; for $N_{pol} < N$, where N is the number of tasks, this means a single policy is based on a mapping with each policy having a number of $\frac{N}{N_{pol}}$ tasks, based on a fixed mapping determined at the start of the lifetime by randomly shuffling tasks, with the number of tasks being as balanced as possible across policies; for $N_{pol} = N$, each policy is used for one single task across the lifetime.

5.1.3 Adaptive assignment of policies to tasks

Here we consider an adaptive assignment of policies to tasks, based on the performance of policies on tasks.

Rationale An adaptive method for policy assignment, based on empirical evidence of the policy's performance on a task, would be useful in the following sense:

- If a policy is assigned to tasks which are incompatible, this strategy could adaptively assign the task to another policy which is able to solve this task.
- Because each policy is tested on all tasks, there is unlikely to be an overfitting, and the policies generated therefore generalise across different tasks and adapt rapidly to new tasks.

Implementation The choice made in this study is to apply traditional exploration-exploitation strategies to select the policy which performs the best most frequently and explore other policies less frequently. A well-known strategy is ϵ-greedy exploration; here the ϵ-greedy exploration is not used for action selection, but it is applied to policy selection, defining a probabilistic map M which selects the best policy with a probability $1-\epsilon$ the best policy and a random policy with probability ϵ . The "best" policy will here be defined as the policy with maximal lifetime average reward velocity:

$$\text{best-policy}(\mathbf{F}_j) = \arg\max_i V_{avg}(\mathcal{P}^i|\mathbf{F}_j)\,, \tag{5.1}$$

where $V_{avg}(\mathcal{P}^i|\mathbf{F}_j)$ is the lifetime reward velocity of policy $\mathcal{P}^i$ on task $\mathbf{F}_j$. The resulting implementation is simple and does not add much computation since the updates are purely based on tracking lifetime performance.

5.1.4 Base learners

The above strategies can easily be applied to a variety of base learners. Here, two base learners, each with different objective functions, are included in the study, one representing value-based reinforcement learners and one representing actor-critic reinforcement learners. Further details on their implementation is given in Section B.2, found in Appendix B.

Deep Recurrent Q-networks (DRQN) This value-based RL method is a recurrent extension to DQN [126] suitable for partially observable environments. Its updates are based on Q-learning [226]. DQN uses the loss function

$$\mathbb{E}_{(o,A,r,o')\sim U(\mathcal{B})}[(r+\gamma\max_{A'} Q(o',A';\hat{\theta}) - Q(o,A;\theta))^2]\,, \tag{5.2}$$

where the experience tuples (o, A, r, o') of observation, action, reward and next action are sampled uniformly from a buffer $\mathcal{B}$, and where a factor γ discounts future rewards. The policy's current parameters are θ whilst another set of policy parameters $\hat{\theta}$, updated infrequently on periodic basis, is used for the target of the neural network. In DRQN, the sampled states are traces of observations which are passed through an LSTM layer such that the system remembers previous time steps. In the present paper's experiments, the buffer is treated as part of the policy, and therefore each policy will have a separate experience buffer.

Proximal Policy Optimisation (PPO) PPO [169] applies a clipped loss function that takes into account that policy updates should not be too large, to allow monotonic improvements:

$$\min(g_t\hat{\mathbb{A}}_t, clip(g_t; 1-\epsilon, 1+\epsilon)\hat{\mathbb{A}}_t)\,, \tag{5.3}$$

where $\hat{\mathbb{A}}_t$ is the advantage estimated at time t by Generalised Advantage Estimation [168], $g_t = \mathcal{P}(A_t|o_t, \theta)/\mathcal{P}(A_t|o_t, \theta_{old})$ is the ratio of the probability of the chosen action A_t according to the new parameters θ and the old parameters θ_{old}, and $clip(a; b, c)$ clips the number a to the interval $[b, c]$. This method is applied in an actor-critic style, in which a critic learns the value of a given observation according to $V(o_t) = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{t+i}]$, with r_t the reward at time t, and in which the actor learns $\mathcal{P}(A_t|o_t, \theta)$, the probability of the chosen action in the observed state o_t, based on the current policy parameters θ.

5.1.5 Estimating task capacity

Analogous to the Vapnik-Chervonenkis dimension [220] providing a quantitative metric for the number of binary classifications a single representation can make, here the above setup with multiple policies is used to construct a metric called the *task capacity*, how many tasks can be represented by a single policy. Here it is proposed to base task capacity on the performance difference obtained by manipulating the amount of policies; base learners with high task capacity are those for which a low number of policies results in a performance close to, or even better than, a one-to-one mapping of tasks to policies.

Task capacity metric Based on the above intuition, one possible metric for the task capacity is:

$$C = N/N^*_{pol}\,, \tag{5.4}$$

where N^*_{pol} is the lowest setting of N_{pol}, the number of policies, which reaches a performance close to, or better than, the one-to-one mapping and N is the total number of tasks. To determine whether the performance is close, an additional tolerance $\epsilon_c \in (0, 1]$ can be specified such that the performance P^* of the N^*_{pol} setting satisfies:

$$P^* \geq (1 - \epsilon_c) * P\,, \tag{5.5}$$

where P is the performance of the one-to-one mapping. Learners with high C and lower ϵ_c therefore can be said to be able to represent C tasks with a loss of at most ϵ_c percentage of P.

Compared to earlier works on capacity [220, 129, 53], this metric is particularly suited to lifelong reinforcement learning scenarios. Furthermore, this contrasts to theory-based approaches which only work for certain topologies and which do not take into consideration the practical limitations inherent to the training procedure and the data distribution [13, 79, 150]. Such practical considerations are especially important in the reinforcement learning setting.

Task selection An important determinant of the above-mentioned task capacity metric is the set of tasks in which it is to be evaluated. To allow interpretation of the task capacity, the task set is defined along a limited number of orthogonal dimensions, spanning a space of tasks which represents the domain of interest. The orthogonal dimensions represent the variable aspects of the decision problem within the domain of interest, such as the dynamics of the environment and the reward function. Consequently, the task capacity metric assesses the number of tasks a policy can learn within the domain of interest.

5.1.6 Other metrics

Aside from task capacity and performance, three other metrics are used to analyse the learners.

Forgetting and transfer Two related metrics are used for assessing forgetting and transfer, both of which assess how earlier tasks affect the initial performance on the current task. In both cases, they are inspired by performance-based metrics which compare the performance of the learner which did not see any previous tasks to the learner which saw previous tasks [23, 205], and forgetting metrics which correct for the random performance on a given task [171]. The resulting metrics allow evaluation in lifelong learning scenarios, are comparable across different tasks, and is suitable for lifelong reinforcement learning as it does do not require multi-task test procedures at the end of each task block. An additional advantage of the metrics is that, when the score on the metrics is compared to score of the task-specific learner which maps one policy to one task, they distinguish between two sources of information, the rate of learning as well as forgetting and transfer.

For forgetting, this leads to the following metric which we will call here the forgetting ratio:

$$\text{forgetting ratio}(t) = \frac{V_{after}(\mathbf{F}(t)) - V_{before}(\mathbf{F}(t))}{V_R(\mathbf{F}(t))}\,, \tag{5.6}$$

where t is the current task block; $V_{after}(\mathbf{F}(t))$ is the current performance on the task $\mathbf{F}(t)$ given during task block t; $V_{before}(\mathbf{F}(t))$ is the performance obtained on the previous block of task $\mathbf{F}(t)$; and $V_R(\mathbf{F}(t))$ is the random performance on the task. By assessing this metric for all the different task-blocks t in which $\mathbf{F}(t)$ was the first presentation of $\mathbf{F}$, the interpretation of the metric is the improvement in performance on task $\mathbf{F}$ by presenting the task again, expressed in units of the random policy's performance on the task.

For transfer learning, this leads to the following metric which we will call here the transfer ratio:

$$\text{transfer ratio}(t) = \frac{V_{after}(\mathbf{F}(t)) - V_R(\mathbf{F}(t))}{V_R(\mathbf{F}(t))}\,, \tag{5.7}$$

where t is the current task block, $V_{after}(\mathbf{F}(t))$ is the performance on the task $\mathbf{F}(t)$ given during task block t, and $V_R(\mathbf{F}(t))$ is the random performance on the task. The metric quantifies, when taking the random performance as a unit, the improvement from a random performance level obtained by seeing prior tasks not equal to $\mathbf{F}(t)$.

Not all of the task transitions are suitable to be evaluated with these metrics. The forgetting metric can only be calculated when a task is seen at least twice in the lifetime, whilst the transfer ratio is defined only for task blocks t in which $\mathbf{F}(t)$ is the first task block that includes $\mathbf{F}$. For each metric, first the valid transitions are obtained, after which the metrics for each $\mathbf{F}(t)$ are then averaged to obtain a single number for each metric.

Policy spread One of the expected benefits of including multiple policies is that each policy can represent a different solution to a task. For adaptive policy changes, an additional benefit is that, if one policy is situated in a region of the parameter space which has low task-performance, then a different policy could be selected from a more favourable region in parameter space. Both factors therefore imply that a diverse set of policies is preferable, to some extent, to maximise the performance of methods that use multiple policies.

To assess the diversity of solutions, the spread in parameter space can be misleading since variability in the parameter space does not necessarily equate to variability in the policies. Instead, as explained in pseudo-code in Algorithm 5.1, here the variability in policies is assessed empirically, based on data obtained by randomly sampling viable observations. The policies then repeatedly generate their output based on the history of observations, and this can then be used to obtain the action probilities, which can then be used to compare the spread of the probability distribution across different policies. In PPO, this is directly based on the output of the actor-network which directly outputs a probability for each action $a \in \mathcal{A}$. In DRQN, this is based on the ϵ-greedy exploration, which first obtains the action $A = \arg\max_A Q(o_t, A)$ and then assigns the probability $1 - \epsilon + \epsilon/|\mathcal{A}|$ for A and the probability $\epsilon/|\mathcal{A}|$ otherwise. The calculation of the spread is based on the total variation distance, a distance metic for probability distributions, applied to all pair-wise combinations of the policies in a condition.

5.2 Experimental set-up

The key purpose of the experiments is to assess how performance is affected as a function of the number of policies, and to assess whether it is beneficial to adaptively learn which policy to assign to which task. To do so, the experiments compare the above-mentioned base learners, as a function of the number of policies included, in a challenging set of lifelong learning environments.

Algorithm 5.1 Calculating the policy spread.

```
sample random observations $o_1, \dots, o_N$, with $N = 10^6$
policy-spread $\leftarrow 0$.
for $i = 1, \dots N - unroll$ do    ▷ account for the trace-length ($unroll$ parameter)
    form traces $h_t \leftarrow \{o_t, \dots, o_{t+unroll-1}\}$
    for $j = 1, \dots, N_{pol}$ do
        compute the probability $\mathcal{P}^i(A_t = A|h_t)$ for all $A \in \mathcal{A}$.
    end for
    $m \leftarrow 0$;    ▷ initialise accumulator
    for $i = 1, \dots, N_{pol}$ do
        for $j = 1, \dots, N_{pol}$ where $j \neq i$ do
            ▷ add the total variation distance between policy $i$ and $j$
            $m \leftarrow m + 0.5 * \sum_{A \in \mathcal{A}} |\mathcal{P}^i(A_t = A|h_t) - \mathcal{P}^j(A_t = A|h_t)|$;
        end for
    end for
    policy-spread $\leftarrow$ policy-spread $+ m/(N_{pol} * (N_{pol} - 1))$;▷ add the average distance
end for
policy-spread $\leftarrow \frac{\text{policy-spread}}{N - unroll}$.    ▷ divide by number of datapoints
```

5.2.1 Lifelong learning environments

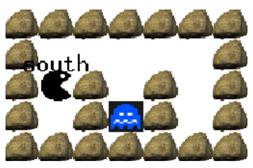
Lifelong learning is a challenge to reinforcement learners because there are a variety of tasks, each with possibly different dynamics, different reward functions, and a different operating environment. In this study, scenarios consisting of 18 tasks are here constructed by manipulating such task characteristics: the dynamics, the reward function, and the topology. For the experiments, small-scale topologies, with small state and observation spaces, are used; this with the aim to limit computational expense and to provide partially observable problems which can be learned within a fairly limited number of learning experiences. The topologies correspond to the cheese-maze [118], Sutton's maze as mentioned in [157] and a 9-by-9 version of the partially observable pacman (POcman) [222], and examples of the resulting tasks are illustrated in Figure 5.1.

Although the scenario involves small-scale topologies, it is challenging in two ways. First, the learner faces many tasks across its lifetime. Tasks follow each other in rapid succession such that, after a single presentation of a task, the learner has not yet converged its parameters. Moreover, because of the variety of tasks, memories of earlier tasks can be lost by the time they are presented again, and a policy learned on one task may transfer negatively to other tasks. Second, the observation is limited, leading to two difficulties: (a) to solve a single task, it is required to learn patterns over the history of observations; (b) a learner cannot immediately infer what is the task from the observation and must therefore can only identify it based on the dynamics of the problem, contrasting to Atari and other video games where a single observation is usually sufficient.

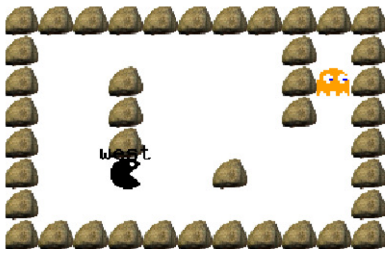
The lifetime of the learner Over its lifetime of 90 million time steps, the learner must navigate through various topologies to maximise cumulative reward by avoiding objects with negative reward and seeking out objects with positive reward, using a fixed set of 5 external actions $\mathcal{A}^E = \{\texttt{north}, \texttt{east}, \texttt{south}, \texttt{west}, \texttt{stay}\}$, and a limited observation consisting of 11 bits of either 1 or -1, which are similar to the partially observable pacman: the first four indicate for each direction in a Von Neumann-neighbourhood whether the position contains an obstacle or not, the next four check for an object in the Von Neumann-neighbourhood, and the final three bits indicate whether or not the object is within a Manhattan distance of 2, 3 or 4 steps from the learner. After each 1000 time steps the learner is reset to the starting location but no terminal states are known to the learner. Then, the learners perform their usual learning cycles again but, in the first $unroll = 15$ time steps, any external actions chosen by the learner is replaced by a random external action. This ensures DRQN's initial memory contains only information relevant to the current episode.

Types of tasks As illustrated in Figure 5.1, the environment's 18 distinct tasks are based on three dimensions, $\mathbf{F} = (reward, dynamic, topology)$, where: $reward \in \{-1, 1\}$ is the reward for touching the main object in the task; $dynamic \in \{0, 1, 2\}$ is a dimension describing how dynamic the main object is, with 0 indicating static, 1 indicating a random step (north, east, south or west) once every 20 time steps, and 2 indicating a policy which reacts defensively for $reward = 1$ or aggressively for $reward = -1.$; $topology \in \{0, 1, 2\}$ the topology of the problems corresponding to the cheese-maze, the Sutton maze and the POcman problem. Behaviourally, when the object is static with $dynamic = 0$, the learner must find a single good location with the object, if $reward = 1$, or without the object, if $reward = -1$; when the object is dynamic with $dynamic > 0$, the learner should either follow the object as closely as possible, if $reward = 1$, or stay away at a safe distance, if $reward = -1$. Unlike the original tasks, objects are never removed when touching them but continue to be a source of reward, until the elementary task ends, and instead of a fixed initialisation, static main objects are randomly chosen from a set of (x, y)-coordinates; see Section B.2 in Appendix B for starting coordinates.

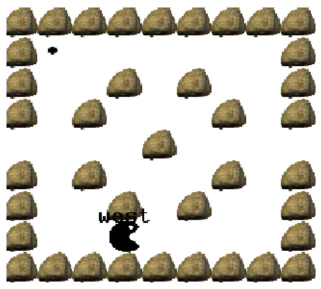
Task sequences As illustrated in Figure 5.2, 18 task sequences were generated by pseudo-randomly sampling, at a frequency of once every 200 000 time steps, a task from a uniform distribution over a set of 18 features. A block in which a same task occurs is called a task block; a single task block has 200 elementary instances of the current task, each consisting of 1000 time steps. To fill all task-time combinations, only the first run's task-sequence is generated randomly while the 17 other sequences have for each task block its task index j computed as $j = (i + n) \mod 18$ where i is the task index of the block in the first task-sequence and n is the task-sequence.



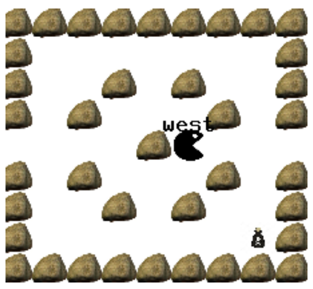
(a) positive, dynamic, cheese maze



(b) negative, dynamic, Sutton maze



(c) positive, static, pacman maze



(d) negative, pacman maze

Figure 5.1: *Illustration of various tasks based on three defining characteristics of the feature vector.*

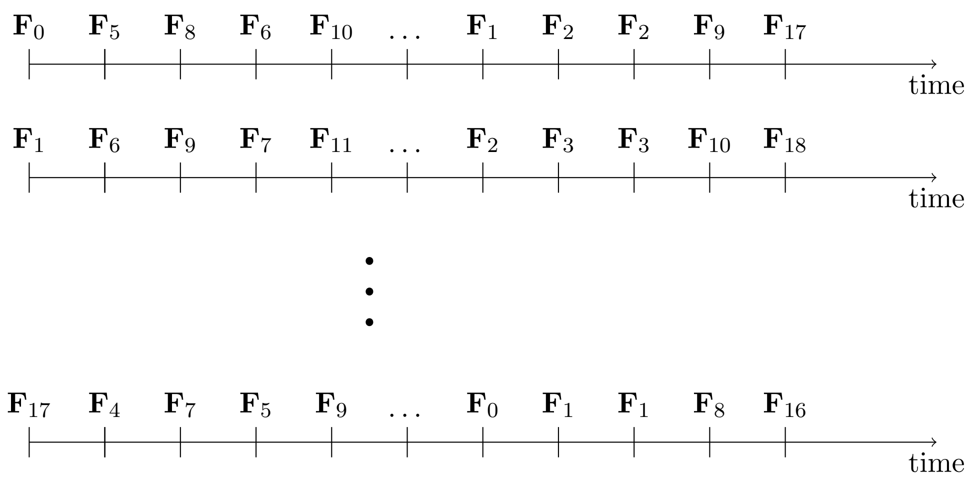

$\mathbf{F}_0$ $\mathbf{F}_5$ $\mathbf{F}_8$ $\mathbf{F}_6$ $\mathbf{F}_{10}$ ... $\mathbf{F}_1$ $\mathbf{F}_2$ $\mathbf{F}_2$ $\mathbf{F}_9$ $\mathbf{F}_{17}$
time
$\mathbf{F}_1$ $\mathbf{F}_6$ $\mathbf{F}_9$ $\mathbf{F}_7$ $\mathbf{F}_{11}$ ... $\mathbf{F}_2$ $\mathbf{F}_3$ $\mathbf{F}_3$ $\mathbf{F}_{10}$ $\mathbf{F}_{18}$
time
$\mathbf{F}_{17}$ $\mathbf{F}_4$ $\mathbf{F}_7$ $\mathbf{F}_5$ $\mathbf{F}_9$ ... $\mathbf{F}_0$ $\mathbf{F}_1$ $\mathbf{F}_1$ $\mathbf{F}_8$ $\mathbf{F}_{16}$
time

Figure 5.2: *Illustration of the task sequences. A total of 18 task features which are sampled randomly to form the first task sequence. The following task sequences are then computed by computing for each block its task index* $j = (i + n) \mod 18$, *where* i *is the task index of the block in the first task-sequence and* n *is the task-sequence.*

5.2.2 Learning systems

To assess the score on the metrics mentioned in Section 5.1.5 and 5.1.6, the base-learners DRQN [69] and PPO [169], as explained in Section 5.1.4, will be subjected to the above-mentioned lifelong learning environments. Each base learner is supplied with an LSTM layer [77] to learn from the previous *unroll* = 15 time steps to allow learning in partially observable environments. These base learners are then embedded in the task assignment strategies mentioned in Section 5.1.2 and 5.1.3, with the adaptive assignment conditions varying in $N_{pol} \in \{2, 4, 9\}$ and the unadaptive varying in $N_{pol} \in \{1, 2, 4, 9, 18\}$; the additional conditions in $\{1, 18\}$ in the unadaptive case are used as baselines which have been used in other previous experiments, with $N_{pol} = 1$ representing the traditional DRQN, and $N_{pol} = 18$ representing the one-to-one mapping of tasks to policies. Parameter settings are mentioned in Section B.2 found in Appendix B.

5.3 Results

The various conditions are analysed in this section. The section is composed of two parts: a performance section which analyses the effect of the number of policies on performance, and a section which explains the performance results using additional statistics. The learned behaviours can be observed on video from the link `https://drive.google.com/drive/folders/1wlf1zh6bhyNhpQvO6siYkX7TTvB1_owR?usp=sharing`.

5.3.1 Performance

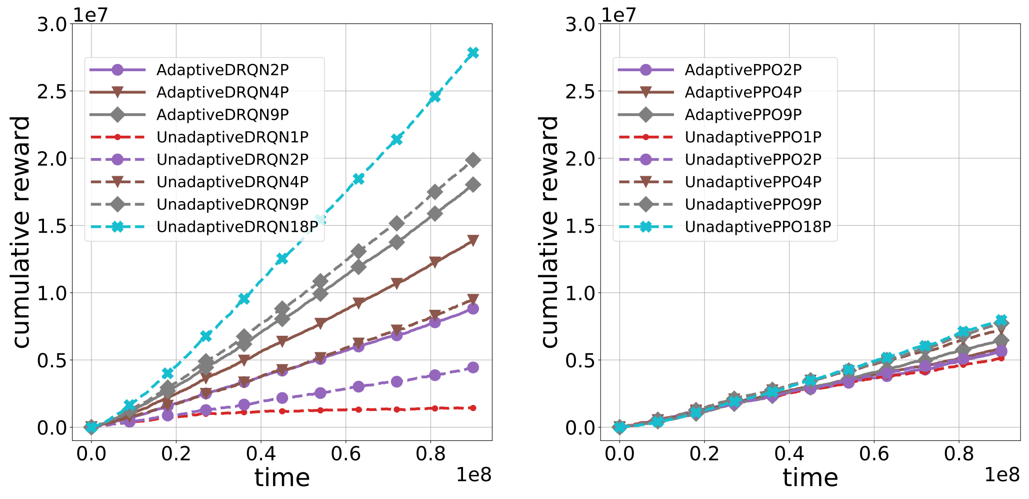
This subsection discusses the performance as a function of the number of policies and the base learner.

Cumulative reward Figure 5.3 illustrates the lifetime average performance of the various learners. Here it can be observed that:

- The cumulative reward intake improves as the number of policies is increased.
- In DRQN conditions, the single policy approach's reward velocity actually *decreases* over time.
- In PPO conditions, all learners are able to improve over the lifetime and the differences between conditions are less pronounced; compared to DRQN, the single policy performance is better and the performances of other conditions are poorer.

Performance metric and statistical significance In Table 5.1, the average and final performance are summarised and analysed in terms of statistical significance. Further, Appendix D includes the ordinal analyses of DRQN and PPO in Table D.1 and Table D.2, respectively.

For DRQN, including multiple policies improved lifetime average performance with a factor 6-20 times, with the strongest effect being observed when comparing the 18-policy learner to the single policy learner. Results on DRQN's lifetime averages show that



(a) DRQN (b) PPO

Figure 5.3: *Cumulative reward depending on the number of policies.*

nearly all differences are significant with significance threshold $\alpha = .05$ and $\alpha = 0.001$, with conditions with more policies outperforming conditions with fewer policies, and the adaptive solutions outperforming the unadaptive solutions with the same number of policies. In case unadaptive algorithms outperformed adaptive algorithms with the same number of policies, the difference was not significant.

For PPO, including multiple policies is also beneficial but has a much smaller effect, and only the 18-policy condition achieves $p < 0.001$; all conditions have a performance of 1-2 times the performance of the single policy condition. The single policy PPO performs much stronger than the single policy DRQN, by a factor 4 for lifetime average performance. In contrast, the multiple policy PPO conditions have a performance 2-3 times lower than DRQN. In PPO conditions, adaptive multi-policy approaches performed worse than their unadaptive counterparts with the same number of policies.

Measuring the final performance by averaging the reward velocity over the final 2 million time steps yields similar results, with two notable exceptions. First, the variability is much higher, and therefore p-values are generally higher. Second, the single policy approach in DRQN deteriorates strongly, and its performance is now down to 0.002. With other final performances in DRQN being in the range of 0.072, for the unadaptive 2-policy, and 0.323, for the 18-policy DRQN approach, this means that the improvement obtained by including multiple policies is between 35 and 133 times in reward velocity, For PPO, the single policy approach has not deteriorated and scores similar to its lifetime average, 0.054.

With an upper bound on optimal performance of 45 million [1], the 18-policy DRQN's

[1]An upper bound for the optimal performance is achieving a reward of 1 in tasks with $reward = 1$ and a reward of 0 in tasks with $reward = -1$. With tasks being equally split in $reward \in \{-1, 1\}$, this implies for a lifetime of 90 million steps an upper bound on the cumulative reward equal to 45 million.

performance just below 30 million is close to optimal considering that a proportion of the time is allocated to exploratory actions.

Table 5.1: *Variance analysis on the effect of adaptivity and the number of policies on performance (mean ± standard deviation). $<$ and $>$ are used to indicate whether the method's performance is higher or lower than its comparison, while p denotes the significance value of the pair-wise F-test. Performance is based on the cumulative reward function as $R(t) = \sum_{\tau=0}^{t} r_\tau$. The* ***lifetime*** *reward velocity is obtained by $R(T)/T$ where $T = 9 * 10^7$ is the total lifetime of the learner, and the* ***final*** *reward velocity is obtained by $\frac{R(T)-R(T-t)}{t}$ where $T = 9 * 10^7$ and $t = 2 * 10^6$. Bold font indicates significance with criterion $\alpha = 0.05$ while * indicates significance with criterion $\alpha = 0.001$.*

(a) DRQN

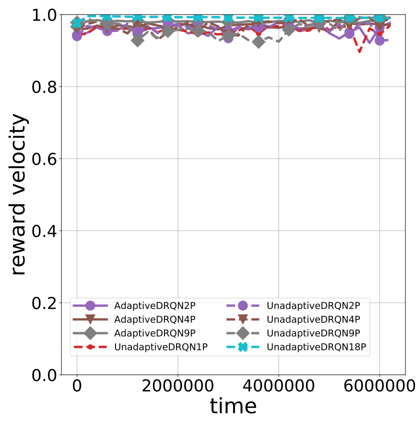
Metric	**Method**	**Performance**	**Comparison**						
lifetime			AdaptiveDRQN4P	AdaptiveDRQN9P	UnadaptiveDRQN1P	UnadaptiveDRQN2P	UnadaptiveDRQN4P	UnadaptiveDRQN9P	UnadaptiveDRQN18P
	AdaptiveDRQN2P	0.098 ± 0.017	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$<, p = 0.532$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	AdaptiveDRQN4P	0.154 ± 0.027	/	$<, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	AdaptiveDRQN9P	0.200 ± 0.025	/	/	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p < 0.001}^*$	$<, p = 0.124$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN1P	0.016 ± 0.007	/	/	/	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN2P	0.049 ± 0.017	/	/	/	/	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN4P	0.105 ± 0.029	/	/	/	/	/	$<, \mathbf{p < 0.001}^{*}s$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN9P	0.220 ± 0.028	/	/	/	/	/	/	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN18P	0.309 ± 0.021	/	/	/	/	/	/	/
final			AdaptiveDRQN4P	AdaptiveDRQN9P	UnadaptiveDRQN1P	UnadaptiveDRQN2P	UnadaptiveDRQN4P	UnadaptiveDRQN9P	UnadaptiveDRQN18P
	AdaptiveDRQN2P	0.104 ± 0.092	$<, p = 0.248$	$<, \mathbf{p = 0.019}$	$>, \mathbf{p = 0.009}$	$>, p = 0.513$	$>, p = 0.909$	$<, \mathbf{p = 0.024}$	$<, \mathbf{p < 0.001}^*$
	AdaptiveDRQN4P	0.160 ± 0.104	/	$<, p = 0.232$	$>, \mathbf{p < 0.001}^*$	$>, p = 0.091$	$>, p = 0.239$	$<, p = 0.199$	$<, \mathbf{p = 0.006}$
	AdaptiveDRQN9P	0.221 ± 0.105	/	/	$>, \mathbf{p < 0.001}^*$	$>, \mathbf{p = 0.006}$	$>, \mathbf{p = 0.023}$	$<, p = 0.802$	$<, p = 0.078$
	UnadaptiveDRQN1P	0.002 ± 0.054	/	/	/	$<, p = 0.091$	$<, \mathbf{p = 0.028}$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN2P	0.072 ± 0.104	/	/	/	/	$<, p = 0.618$	$<, \mathbf{p = 0.008}$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN4P	0.098 ± 0.108	/	/	/	/	/	$<, \mathbf{p = 0.026}$	$<, \mathbf{p < 0.001}^*$
	UnadaptiveDRQN9P	0.236 ± 0.135	/	/	/	/	/	/	$<, p = 0.178$
	UnadaptiveDRQN18P	0.323 ± 0.124	/	/	/	/	/	/	/

(b) PPO

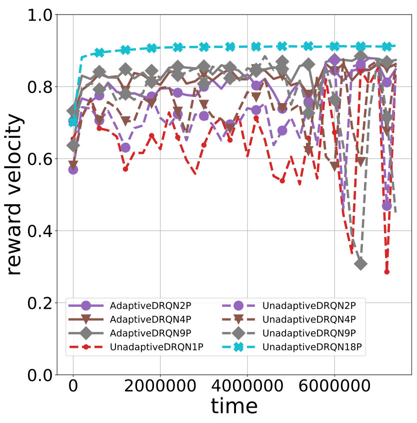
Method	**Performance**		**Comparison**						
lifetime			AdaptivePPO4P	AdaptivePPO9P	UnadaptivePPO1P	UnadaptivePPO2P	UnadaptivePPO4P	UnadaptivePPO9P	UnadaptivePPO18P
	AdaptivePPO2P	0.062 ± 0.012	$<, p = 0.607$	$<, p = 0.178$	$>, p = 0.341$	$<, p = 0.878$	$<, \mathbf{p = 0.013}$	$<, \mathbf{p = 0.003}$	$<, \mathbf{p < 0.001}^*$
	AdaptivePPO4P	0.065 ± 0.011	/	$<, p = 0.343$	$>, p = 0.139$	$>, p = 0.743$	$<, \mathbf{p = 0.032}$	$<, \mathbf{p = 0.008}$	$<, \mathbf{p = 0.001}$
	AdaptivePPO9P	0.072 ± 0.017	/	/	$>, \mathbf{p = 0.036}$	$>, p = 0.243$	$<, p = 0.278$	$<, p = 0.102$	$<, \mathbf{p = 0.038}$
	UnadaptivePPO1P	0.057 ± 0.011	/	/	/	$<, p = 0.296$	$<, \mathbf{p = 0.001}$	$<, \mathbf{p < 0.001}^*$	$<, \mathbf{p < 0.001}^*$
	UnadaptivePPO2P	0.063 ± 0.013	/	/	/	/	$<, \mathbf{p = 0.022}$	$<, \mathbf{p = 0.005}$	$<, \mathbf{p < 0.001}^*$
	UnadaptivePPO4P	0.081 ± 0.017	/	/	/	/	/	$<, p = 0.542$	$<, p = 0.325$
	UnadaptivePPO9P	0.086 ± 0.018	/	/	/	/	/	/	$<, p = 0.751$
	UnadaptivePPO18P	0.088 ± 0.015	/	/	/	/	/	/	/
final			AdaptivePPO4P	AdaptivePPO9P	UnadaptivePPO1P	UnadaptivePPO2P	UnadaptivePPO4P	UnadaptivePPO9P	UnadaptivePPO18P
	AdaptivePPO2P	0.055 ± 0.081	$<, p = 0.738$	$<, p = 0.723$	$>, p = 0.987$	$<, p = 0.717$	$<, p = 0.537$	$<, p = 0.564$	$<, p = 0.243$
	AdaptivePPO4P	0.070 ± 0.098	/	$>, p = 1.000$	$>, p = 0.737$	$<, p = 0.985$	$<, p = 0.800$	$<, p = 0.795$	$<, p = 0.441$
	AdaptivePPO9P	0.070 ± 0.088	/	/	$>, p = 0.723$	$<, p = 0.984$	$<, p = 0.789$	$<, p = 0.787$	$<, p = 0.418$
	UnadaptivePPO1P	0.055 ± 0.090	/	/	/	$<, p = 0.717$	$<, p = 0.544$	$<, p = 0.568$	$<, p = 0.256$
	UnadaptivePPO2P	0.071 ± 0.094	/	/	/	/	$<, p = 0.811$	$<, p = 0.806$	$<, p = 0.443$
	UnadaptivePPO4P	0.082 ± 0.093	/	/	/	/	/	$<, p = 0.973$	$<, p = 0.594$
	UnadaptivePPO9P	0.084 ± 0.116	/	/	/	/	/	/	$<, p = 0.658$
	UnadaptivePPO18P	0.107 ± 0.096	/	/	/	/	/	/	/

Task performances To visualise how the different learners perform on the different kinds of tasks, Figures 5.4 and 5.5 show performances of the learners specific to tasks that represent the extremes: $(reward = -1, dynamic = 0, topology = 2)$ represents the easiest task for negative reward environments, since the object of interest is static and the topology is large and cluttered, meaning it is difficult to encounter the object by chance; $(reward = -1, dynamic = 2, topology = 0)$ represents the most difficult task for negative reward environments, since the object is highly dynamic in a small topology where it is difficult to escape; $(reward = 1, dynamic = 0, topology = 0)$ represents the easiest positive reward task since once the object is found in a small topology, the agent simply can stay put on that location; $(reward = 1, dynamic = 2, topology = 2)$ represents the most difficult positive reward task since the agent must continually chase the highly dynamic object in a large cluttered environment.

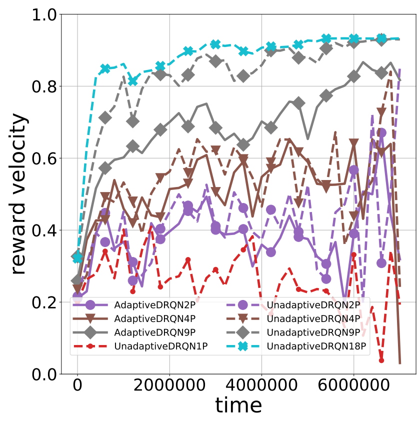
For DRQN, a general observation is that $N_{pol} < 9$ often leads to a degrading task performance over time, whilst the 9- and 18-policy solutions are increasing towards a nearly identical final task performance. However, for task $(-1, 2, 0)$ it can be observed that initially the undaptive 9-policy solution appears to learn stably, but then later has declining performance, and that the adaptive 9-policy solution avoids this problem. This suggests the use of adaptivity as a means to overcome catastrophic forgetting.



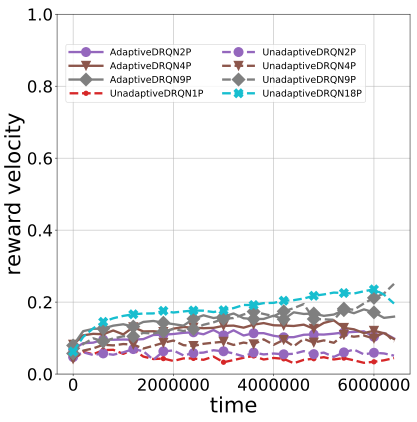
(a) $\mathbf{F} = (-1, 0, 2)$



(b) $\mathbf{F} = (-1, 2, 0)$



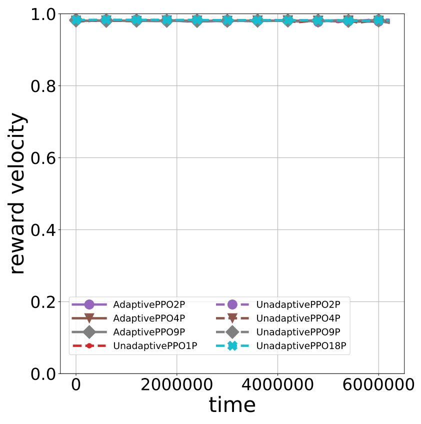
(c) $\mathbf{F} = (1, 0, 0)$



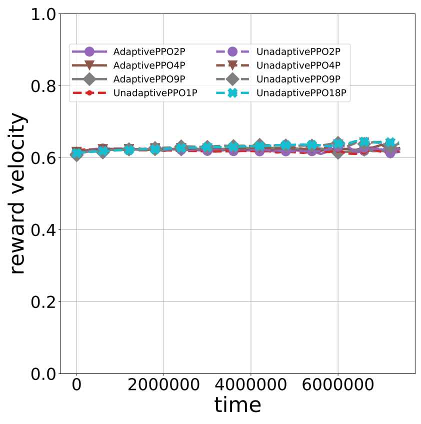
(d) $\mathbf{F} = (1, 2, 2)$

Figure 5.4: *DRQN performance development on individual tasks, depending on the number of policies. Since the lifelong scenario involves different blocks of tasks, the different blocks of each unique task are here joined together, illustrating the total time spent in the task on x-axis, and the performance on that task on y-axis. The task is indicated in the subcaption as* $\mathbf{F}$*, with values along three dimensions: the reward incurred by the object* $\{-1, 1\}$*, the dynamicity of the object* $\{0, 1, 2\}$*, and the topology of the environment* $\{0, 1, 2\}$*. Performance is the reward velocity normalised such that 0 indicates worst and 1 indicates best possible performance.*

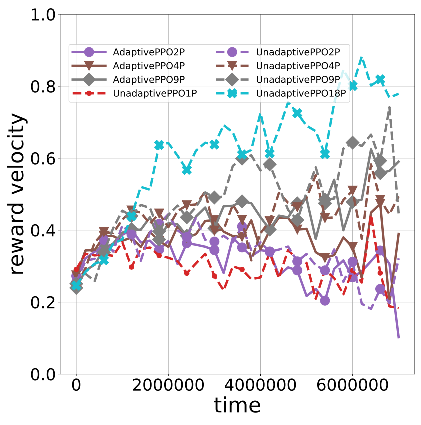
For PPO, static tasks appear to be learned more slowly, and in dynamic tasks, PPO struggles to learn anything. In general, few differences can be observed between the different settings of N_{pol}, except that the $N_{pol} = 18$ setting develops more smoothly over time, due to there being no interfering task blocks for any given policy.



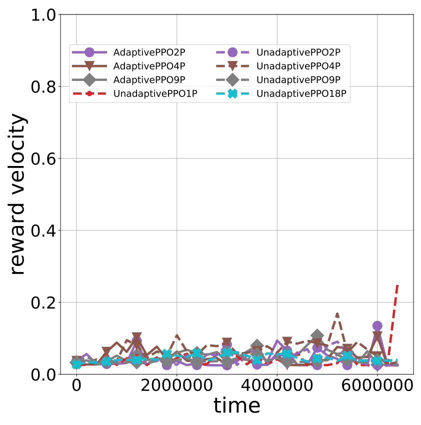
(a) $\mathbf{F} = (-1, 0, 2)$



(b) $\mathbf{F} = (-1, 2, 0)$



(c) $\mathbf{F} = (1, 0, 0)$



(d) $\mathbf{F} = (1, 2, 2)$

Figure 5.5: *PPO performance development on individual tasks, depending on the number of policies. Since the lifelong scenario involves different blocks of tasks, the different blocks of each unique task are here joined together, illustrating the total time spent in the task on x-axis, and the performance on that task on y-axis. The task is indicated in the subcaption as $\mathbf{F}$, with values along three dimensions: the reward incurred by the object $\{-1, 1\}$, the dynamicity of the object $\{0, 1, 2\}$, and the topology of the environment $\{0, 1, 2\}$. Performance is the reward velocity normalised such that 0 indicates worst and 1 indicates best possible performance.*

5.3.2 Explanatory results

To explain the above findings in greater detail, the role of adaptivity and the effect of prior tasks is investigated more closely.

The effect of policy spread One of the expected benefits of multiple policy approach is that if a particular policy is situated in a bad region of parameter space, then it is easier to escape this region simply by selecting one of the alternative policies, which may be located in more favourable locations of parameter space. If this is indeed true, then one would expect that multiple policies are more beneficial when the policy spread is high, and especially when they are chosen adaptively since that allows to select one of the alternative policies. If this hypothesis is true, then it can be expected that: PPO has low policy spread, since the effect of multiple policies is weak and there is no benefit of adaptivity, and, that DRQN has high policy spread, since for DRQN adaptive learners outperform their unadaptive counterparts, with the exception of the 9-policy case where the difference is non-significant. Figure 5.6 shows that this is indeed the case. For adaptive PPO methods, all conditions have a policy spread within $[0.1, 0.3]$, whilst for adaptive DRQN methods, policy spread lies within $[0.6, 0.7]$.

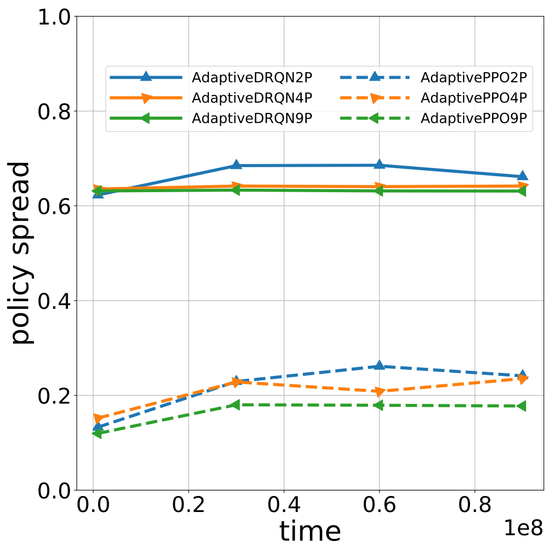


Figure 5.6: *Policy spread as a function of time, depending on the number of policies.*

Task capacity Normally, one would expect that performance increases after a successive presentation of the same type of task; however, catastrophic forgetting could prevent this from happening. Using the forgetting ratio from Equation 5.6, Figure 5.7 illustrates that DRQN learners with a low number of policies fail to maintain the performance on a particular task across the lifetime, especially with many interfering task blocks. For the single policy approach, one interfering task block appears to be enough to cause forgetting. In contrast, PPO methods all have a forgetting ratio similar to the 18-policy approach, indicating they do not forget.

Similarly, general learners are expected to be able to transfer their knowledge learned in earlier tasks to a completely new unseen task. Figure 5.7 illustrates, using the transfer ratio from Equation 5.7 , that most PPO methods score well above their 18-policy counterpart, indicating a larger transfer among tasks; meanwhile, the DRQN methods score well below the 18-policy DRQN condition.

When the 18-policy solutions are compared between DRQN and PPO, the transfer and forgetting ratio scores show that, regardless of any transfer or forgetting, DRQN learns more rapidly than PPO. This in fact may be one of the explanations for the results; rapid learning on one task may improve task-specific policies but due to overfitting may result in negative transfer and catastrophic forgetting when more than one task has to be learned.

The main results on forgetting and transfer explain the difference in *task capacity* between PPO and DRQN: DRQN is strongly affected by manipulating the number of policies because it is prone to forgetting and does not transfer well between tasks; in contrast, PPO can represent more tasks without large performance effects. When the task capacity metric in Equation 5.4 is applied to the lifetime average performance with a setting of $\epsilon_c = 40\%$, this results in DRQN having a task-capacity $C = 2$ and PPO having a task capacity of $C \geq 18$. This can be seen in Table 5.1, where: UnadaptiveDRQN9P has a lifetime average of .220, a performance of 71.1% of the UnadaptiveDRQN18P performance of 0.309; UnadaptivePPO1P has a performance of 0.057, which is 64.8% of the UnadaptivePPO18P. With $\epsilon_c = 0$, both PPO and DRQN have a task capacity $C = 1$, since the 18-policy conditions obtain the highest performance.

5.3.3 Resource consumption

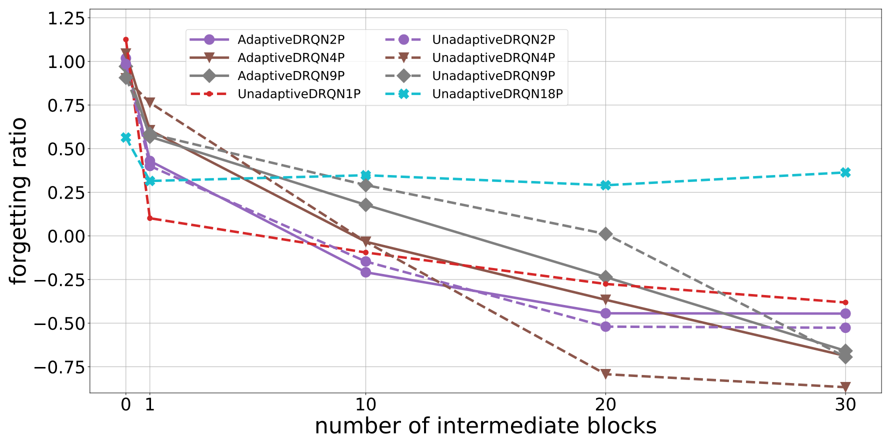
Illustrative of the need for learning with a limited number of policies, an increase in memory consumption is observed as one includes more policies. Experiments are conducted on the IRIDIS4 supercomputer [215] using a single Intel Xeon E5-2670 CPU (2.60GHz) with a varying upper limit to RAM: 1- and 2-policy conditions use 4GB; 4-policy conditons use 8GB; 9-policy conditions use 16GB; 18-policy conditions use 28GB. 6-9 days of computing are required, with consumption lowering as RAM is increased.

5.4 Discussion

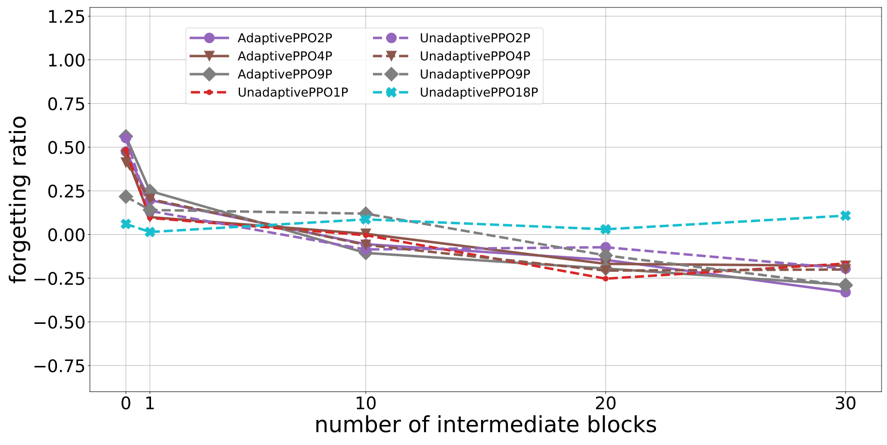
5.4.1 Strengths

This chapter shows that using a more limited number of policies can achieve great performance gains compared to a single policy approach. Similar to the experiments in this thesis, using multiple policies would yield the strongest performance advantages when task sequences involve (i) a rapid succession of different tasks; (ii) distinct task clusters each of which allow significant positive transfer within clusters and negative transfer between clusters. It is further expected that using multiple representations and similar adaptive strategies are applicable to supervised learning tasks with only minor modifications.

The study illustrates a novel type of analysis based on *task capacity*, the number of tasks a single base learner may represent. Due to combining both the acquisition and storage of new information, task capacity depends on both the learning algorithm and the architecture, and therefore gives an indication of the overall lifelong learning scalability



(a) DRQN

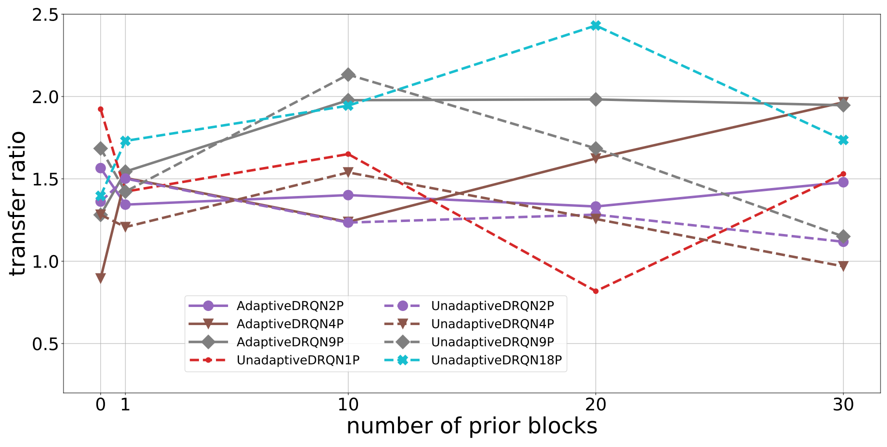


(b) PPO

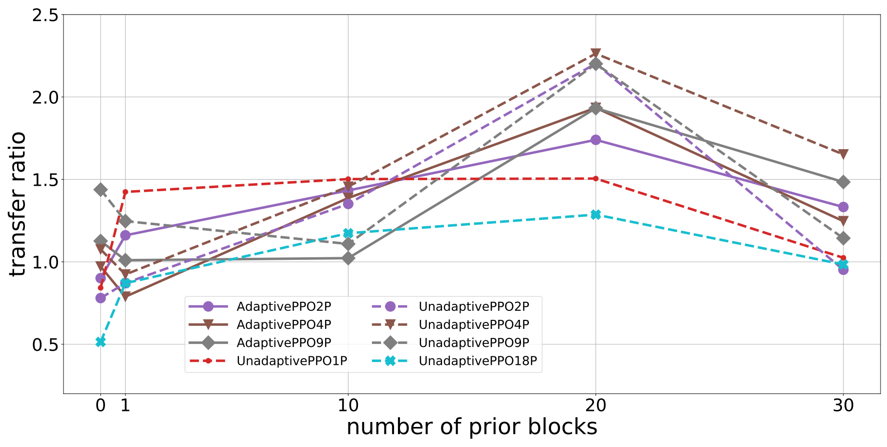
Figure 5.7: *Effect of intermediate tasks on forgetting depending on the number of policies. The x-axis shows the number of interfering tasks, measured in bins; for example, $x = 1$ means a number of interfering tasks in $[1, 9]$ and the final value of $x = 30$ means 30 or greater. The y-axis shows the forgetting ratio as defined in Equation 5.6, with the interpretation being that, $y = -1$ means a decrease of 1 time the random performance, whilst $y = 1$ means an increase of 1 time the random performance, when compared to the performance of the same task in a task block before the interfering task blocks.* ***Note:*** *the 18-policy condition (cyan) can be seen as a baseline since there is no catastrophic forgetting possible because each of its policies sees only a single task; therefore every learner below the 18-policy will have some forgetting.*

of a base learner for a given domain. In this sense it contrasts to earlier metrics which isolate particular aspects relevant to lifelong learning such as transfer and catastrophic forgetting [171, 95, 43, 114, 92, 205]. The task capacity analysis is, however, easily supplemented with additional analyses in terms of transfer and forgetting metrics to provide further insights into the mechanisms behind a base-learner's task capacity.

The results show PPO and DRQN differ strongly in their task capacity: allowing a drop of 40% in performance, PPO is able to represent 18 tasks with just 1 policy, whilst a single DRQN policy can learn only 2 tasks. PPO is shown to be only weakly affected by the number of policies; a possible explanation is that PPO's objective function stimulates



(a) DRQN



(b) PPO

Figure 5.8: *Effect of prior tasks on transfer depending on the number of policies. The x-axis shows the number of prior tasks, measured in bins; for example, $x = 1$ means a number of interfering tasks in $[1, 9]$ and the final value of $x = 30$ means 30 or greater. The y-axis shows the forgetting ratio as defined in Equation 5.7, with the interpretation being that, as a result of the prior task blocks, $y = -1$ means a decrease of 1 time the random performance, whilst $y = 1$ means an increase of 1 time the random performance, when compared to the random performance.* ***Note:*** *the 18-policy condition (cyan) can be seen as a baseline since there is no transfer learning possible because each of its policies sees only a single task; therefore every learner above the 18-policy will have some transfer learning.*

monotonic improvement over time, while DRQN policies are overfitting on new patterns allowing resulting in a low transfer to new tasks and rapid forgetting of learned tasks. This explanation is supported by the difference in transfer and forgetting metrics and the decreasing performance of the single-policy DRQN over time.

Another key result is that a simple performance-based strategy to adaptively assign policies to tasks can improve performance, likely because the time to learn the task from a bad location in parameter space is longer than the time to experiment with various policies. For PPO this is not found, and it is suggested this is because PPO's policies' performances are close to each other, and searching over policies simply wastes experience for learning a single good policy, with PPO being known as a slower but monotonically improving learner. This is indeed supported by the policy spread metric, which shows that the responses of the different policies are much more dissimilar for DRQN than for PPO. The adaptive strategy appears to be beneficial not only in the early stages of learning, when selecting a policy might yield more rapid performance advantages compared to learning it, but also to switch to another policy in the later stages of learning when experience with interfering tasks has decreased performance on a previously learned task.

Finally, the study is not simply an empirical study but also contributes a novel metric. Due to different base learners differing in how much tasks they can represent, learning with multiple policies can serve as a metric for task capacity of lifelong learning algorithms, analogous to the Vapnik-Chervonenkis dimension [220] for classification algorithms. To continue this reasoning more rigorously, theoretical frameworks could be developed based on a similar notion of task capacity. A more direct use of the metric here proposed for task capacity is to allow it to select how many policies are required for learning a set of tasks most efficiently. Similarly, the task capacity can be seen as the "goodness of a lifelong learner": the more tasks a policy is able to represent, the stronger the lifelong learning aspects of this learner, particularly in regards to transfer learning and forgetting. An important issue to assess task capacity in this empirical way is the choice of the tasks; here it is suggested to perform the capacity test on the target domain, in which the user wants to apply the learning system, whilst manipulating those task parameters that can reasonably be expected to vary and affect performance.

5.4.2 Limitations

Although the present study shows a large performance benefit in using multiple policies, the approach taken in this study has its limitations when used as a method for lifelong learning, as opposed to an analysis tool. First, there is currently a requirement to provide the task to help select the policy; task identification methods such as the flat forget-me-not strategy [95] could be used to overcome this limitation. Second, the adaptive policy assignment strategy relies on performance metrics and policy exploration algorithms, which may not be optimal; future work could investigate a variety of policy selection strategies to gain further performance improvements. Third, here it was assumed that policy intervals started at the start of a new episode; however, the environment may be inherently non-episodic or the knowledge of resets may be unavailable to the learner. To solve this, an additional learning mechanism could be implemented to determine the start of policy intervals. Fourth, in unknown environments, the designer has limited knowledge of the tasks, and therefore the choice of the tasks may not be representative for the target application. Finally, because the number of tasks representable by a single base learner is limited, scenarios with a vast number of tasks may require too much memory consumption due to the many base learners. As a remedy to the increasing memory requirements and the performance loss observed compared to using task-specific policies, the use of a limited number of policies could be combined with other lifelong learning strategies; this would be useful to maintain a diversity of policies, each of which specialise on a large set of tasks, whilst making each policy as robust as possible to variations in that specialisation. For example, methods which apply a single parameter vector for multiple tasks combined with task-specific parameters [42, 110] may be applied, one to each of the subsets of tasks.

5.5 Summary

In long-term unknown environments, different tasks may be presented in sequence, requiring lifelong learning. This chapter empirically investigates the use of multiple policies, each specialised on a subset of tasks. To improve performance in lifelong learning scenarios and to analyse different base reinforcement learners' task capacity, using a study which manipulates the number of policies included. Results on an 18-task lifelong learning scenario, including DRQN and PPO as base-learners, show that (a) allowing a 40% loss of performance, single-policy DRQN can learn at most 2 tasks, whilst PPO can learn all 18 tasks included in the study; (b) DRQN's performance can greatly exceed that of PPO when using 2 or more policies; (c) adaptively learning to assign policies to tasks can improve the performance up to a factor of 2, but only for the DRQN learner. The key contribution of this chapter is to provide a novel learning system and novel analysis tools for lifelong learning, both of which are based on policies specialising on a subset of the tasks.

Chapter 6

Towards active adaptive perception in lifelong learning environments

In long-term unknown environments, a learner may be confronted with the challenge of learning a set of tasks presented sequentially across the lifetime. In such lifelong learning environments, the same task learned earlier in the lifetime can be unlearned, due to catastrophic forgetting: when specialising on novel tasks, the parameters learned for an earlier task may be completely overridden.

This is an issue that is currently not taken into account in the active adaptive perception learners mentioned in Chapter 4. This is because the evaluation module of the generic architecture is implemented as the Success Story Algorithm. Although a previous study in [166] showed that SSA agents could transfer their knowledge across a hand-crafted curriculum consisting of similar tasks of increasing complexity, such a curriculum is not always possible and in more challenging lifelong learning scenarios, SSA may have strong limitations. Due to the complete removal of self-modifications which do not currently improve the lifetime reward velocity, it appears plausible that such learners are prone to catastrophic forgetting when there is a sudden reward functions change.

For the above reason, this chapter investigates the limitations of SSA when applying active adaptive perception learners in lifelong learning environments, and proposes a novel long-term utility function which corrects for the different reward functions that occur in different tasks. Due to the novel utility function, this also implies further changes to SSA, resulting in the Lifelong Success Story Algorithm.

6.1 Lifelong Success Story Algorithm

The active adaptive perception learners presented in Chapter 4 all rely on the Success Story Algorithm as their evaluation module. This section illustrates the limitations of SSA and proposes mechanisms to overcome these limitations. Reformulating SSA's stack, which stores the valid self-modifications, plays an essential role in overcoming these limitations; therefore, the section makes extensive use the operators and data structures from Table 3.1.

6.1.1 Rationale

The Success Story Algorithm maintains only self-modifications which ensure a lifetime reward acceleration. This implies that, when different tasks with different reward function are presented sequentially, this can lead to a false impression of improvement or performance decline. As illustrated in Figure 6.1, when negative rewards occur after a history of success, this can completely remove any earlier learned knowledge, even if this is close to the new task's optimal performance level, resulting in *catastrophic forgetting and failure to learn*. In similar spirit, when the new task is highly rewarding compared to previous tasks, a new task can give the false impression of improvement, leading the agent to learn poor behaviours, in what may be called *overly eager learning*.

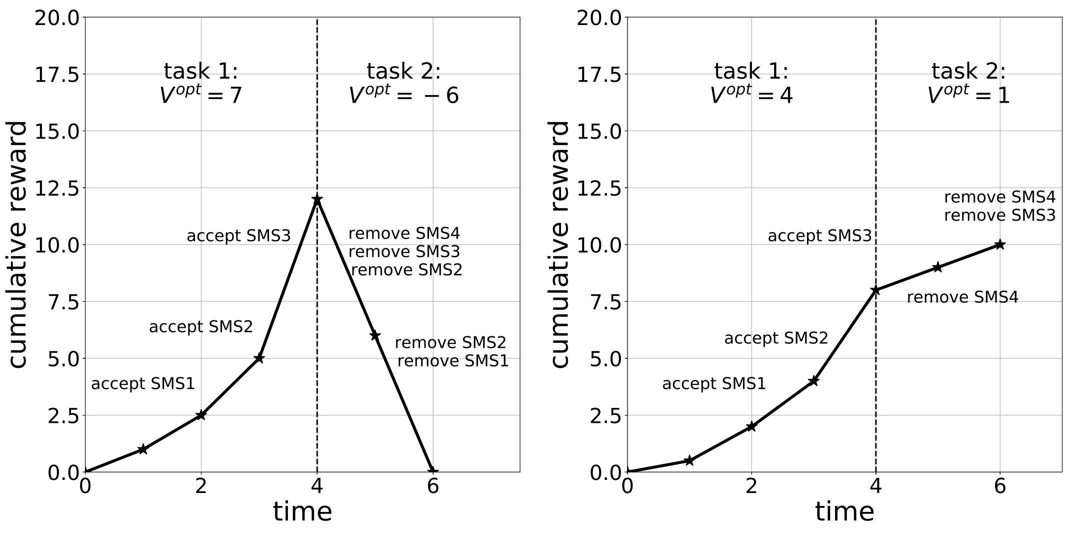
6.1.2 Correcting for different reward functions: the Lifelong Success Story Criterion

The above provides a rationale to provide SSA with a correction for the different reward functions that may occur during the active adaptive perception learner's lifetime. When in lifelong learning different tasks each are characterised by different reward velocity profiles, one way to avoid the problems mentioned in Section 6.1.1 is to account for all the different *task-specific reward velocities*. This idea is illustrated in Figure 6.2.

Based on such task-specific velocities, a novel utility function called the *baseline-adjusted global velocity*, or $\tilde{V}$ for short, is proposed to improve evaluation of active adaptive perception learners in lifelong learning scenarios. This metric forms the basis of the *Lifelong Success Story Criterion* :

$$Firsts.top() = \operatorname*{arg\,max}_{e \in Firsts \subset \mathscr{S}} \tilde{V}(e) = \sum_{i=1}^{N} w_i (V_{\mathbf{F}_i}(e.t_{\mathbf{F}_i}) - B_i)\,, \tag{6.1}$$

where $Firsts$ is a subset of the stack which includes only the starts of self-modification sequences; $Firsts.top()$ is the entry in the stack which started the last self-modification



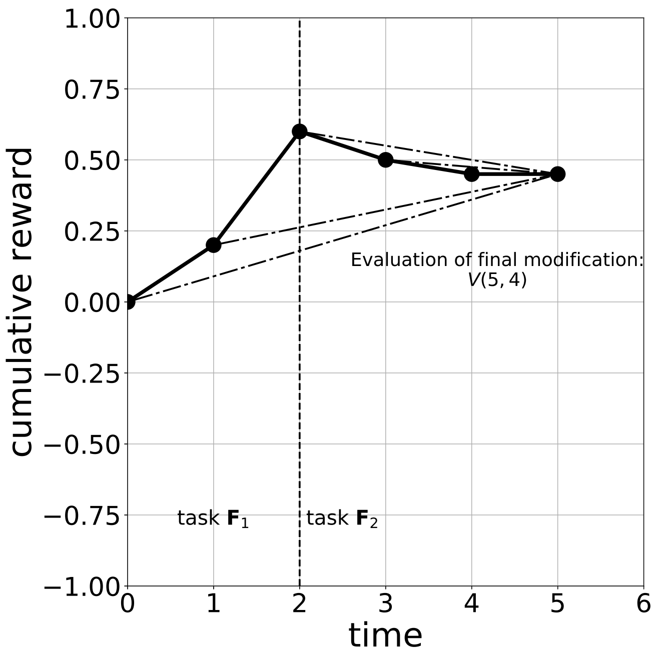
(a) Reward depression (b) Reward speed decline

Figure 6.1: *An illustration of catastrophic forgetting in SSA, based on hypothetical data. For simplicity it is assumed checkpoints are spaced evenly across time. (a) When confronted with a task 2 with a lower optimal performance of $V^{opt} = -6$ when compared to the earlier task with $V^{opt} = 7$, even an optimal task performance may result in all self-modification sequences (SMS) being removed. This removes any knowledge learned in both the current and the earlier task. (b) Even when cumulative reward is increasing, transitioning from a task with high optimal performance, $V^{opt} = 4$, to a task with a lower optimal performance, $V^{opt} = 1$, may result in knowledge removal about the current task and parts of the previous task.*

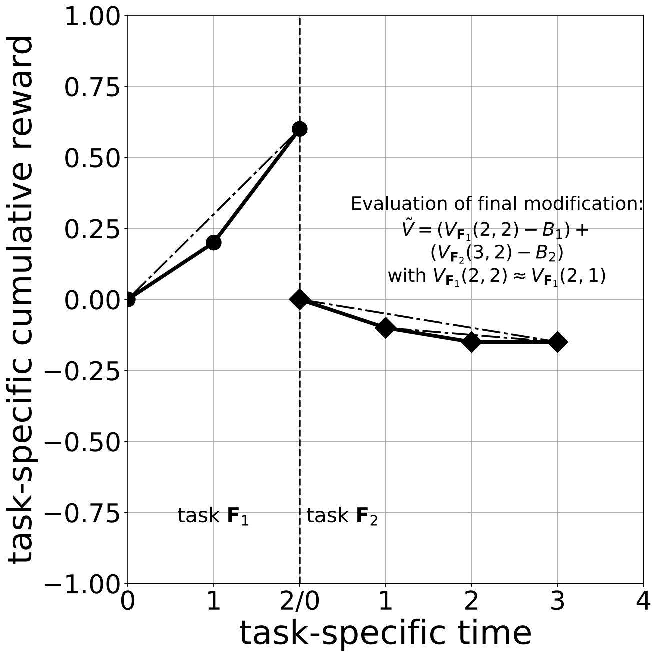
sequence; $V_{\mathbf{F}_i}(e.t_{\mathbf{F}_i}) = \frac{R_{\mathbf{F}_i} - e.R_{\mathbf{F}_i}}{t_{\mathbf{F}_i} - e.t_{\mathbf{F}_i}}$ calculates the task-specific reward velocity for the interval between the moment e was pushed onto the stack $\mathscr{S}$, corresponding to $e.R_{\mathbf{F}_i}$ and $e.t_{\mathbf{F}_i}$, and the current moment, corresponding to $e.R_{\mathbf{F}_i}$ and $e.t_{\mathbf{F}_i}$, and the current moment, corresponding to $R_{\mathbf{F}_i}$ and $t_{\mathbf{F}_i}$; $w_i \in \mathbb{R}^+$ can be set to reflect the task's importance and the distribution of tasks; and, where B_i is the velocity baseline, a running average of the $V_{\mathbf{F}_i}$ observed across the lifetime.

The introduction of B_i into Equation 6.1 is crucial because it allows the Lifelong SSA evaluations to correct for the different tasks' baseline velocities, and thereby overcomes the aforementioned problems of catastrophic forgetting and eager learning (see Section 6.1.1).

Equation 6.1 incorporates two aspects that are different from the traditional Success Story Criterion. A first factor is the utility function $\tilde{V}$ and the procedures needed to compute it. A second factor is the arg max operator: in traditional SSA there is only the requirement to check the top two entries in the stack, but this strategy does not work for $\tilde{V}$. These two aspects of the Lifelong Success Story Algorithm are discussed in the next two sections.



(a) Global velocity



(b) Task-specific velocities

Figure 6.2: *Illustration of task-specific reward velocities, based on hypothetical data with each data point representing the start of a self-modification sequence. Panel (a) illustrates the global velocities used in traditional SSA. Panel (b) illustrates the task-specific velocities, calculated by dividing the accumulated task-specific reward with the task-specific time that has passed. The annotation in (b) also illustrates that when no time has passed, the entry is unevaluated on* $\mathbf{F}_1$ *and therefore the task-specific velocity is estimated by taking the most recent observed velocity for that task,* $V_{\mathbf{F}_1}(2,2) \approx V_{\mathbf{F}_1}(2,1)$.

6.1.3 Computing the baseline-adjusted global velocity

To compute the baseline-adjusted global velocity, there are two steps: computing task-specific velocities and combining the different task-specific velocities to a single metric.

Computing task-specific velocities To be able to compute the task-specific velocities, the stack structure used in traditional SSA must be modified to include data in a task-specific manner.

The stack, denoted as $\mathscr{S}$, contains different entries $e \in \mathscr{S}$, each of which represent two types of information: (a) information to restore the learner, such as the parameters $\mathcal{X}$ of the instruction module or perception module obtained just before a self-modification or perceptual modification, and the address *address* at which the self-modification was made; and (b) auxiliary information to be able to calculate the performance of the learner since the introduction of the modification, such as the time t and cumulative reward R recorded at the time that a modification was made.

In traditional SSA, the performance of a self-modification sequence is evaluated by obtaining the first entry e in the self-modification sequence, and then computing $\frac{R(t')-e.R}{t'-t}$. In Lifelong SSA, the auxiliary information in the stack must be modified because the Lifelong Success Story Criterion is based on the baseline-adjusted global velocity $\tilde{V}$, which requires calculation of *task-specific* reward velocities, SSA is modified such that entries $e \in \mathscr{S}$ now have task-specific timers $e.t_{\mathbf{F}_i}$ and task-specific cumulative rewards $e.R_{\mathbf{F}_i}$ for all N tasks seen so far. To do so, Lifelong SSA keeps track of $t_{\mathbf{F}_i}$ and $R_{\mathbf{F}_i}$ for all tasks seen so far, $\mathbf{F}_1, \dots, \mathbf{F}_N$, and, whenever a new stack entry is pushed, it then sets $e.t_{\mathbf{F}_i} \leftarrow t_{\mathbf{F}_i}$ and $e.R_{\mathbf{F}_i} \leftarrow R_{\mathbf{F}_i}$.

Recording task-specific performance data does not yet help solve all issues; as illustrated in Figure 6.2, for many tasks no task-specific time will have passed in between the start of a self-modification sequence and the evaluation time, and computing the velocity for those tasks is not possible. To solve this problem, Algorithm 6.1 estimates unevaluated velocities based on the most recent prior entry evaluated for $\mathbf{F}_i$. This is illustrated in Algorithm 6.1. This procedure ensures such entries cannot be removed without evidence and, once task time passed, their task velocities are calculated as usually. When a task is not seen at all, it does not contribute to $\tilde{V}$; the system therefore remains open to new tasks at all times.

Combining task-specific velocities into a single metric After calculating the task-specific velocities, these then have to be combined into a single metric $\tilde{V}$. One can see from Equation 6.1, that there is some freedom to choose the possible weightings. As natural weighting of the tasks is to set 1 for all the seen tasks, and 0 for all unseen tasks. The main reason for selecting this task weighting in the coming experiments is to reduce

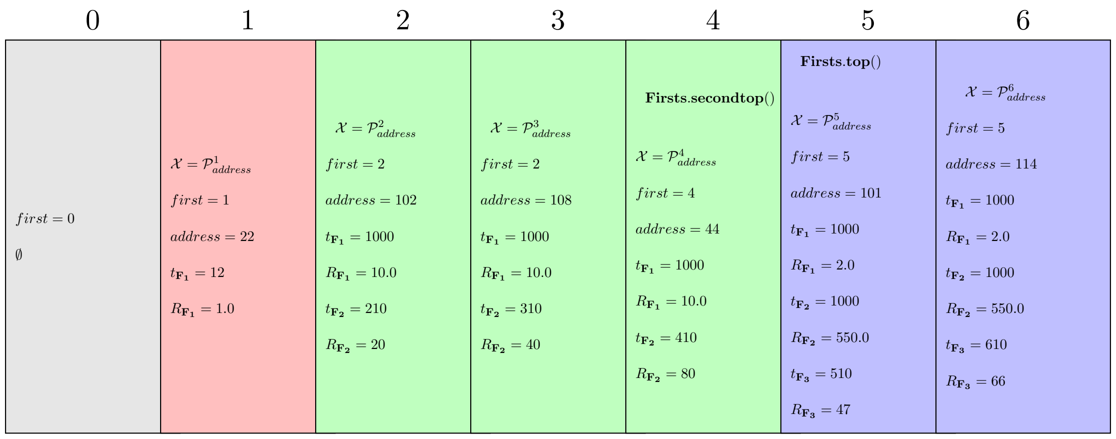


Figure 6.3: *Illustration of the stack structure to record self-modifications, for a sequence of three consecutive tasks* $\mathbf{F}_1$ *(red),* $\mathbf{F}_2$ *(green) and* $\mathbf{F}_3$ *(blue), each ran for a total of 1000 time steps. The stack entries include many of the properties included in the traditional SSA stack:* **first***, the index of the first entry in stack of the self-modification sequence;* $\mathcal{P}^{\mathbf{t}}$*, the old policy at the time* t *when the self-modification was performed, needed to recover the old policy when its corresponding self-modification sequence is popped;* **address***, the address at which* $\mathcal{P}^{old}$ *was modified. However, instead of a global timer, each entry has a separate timer* $\mathbf{t_F}$ *and cumulative reward* $\mathbf{R_F}$ *for each task* $\mathbf{F}$ *seen so far. As in traditional SSA, the initial entry (grey) cannot be popped off, and represents the initial stack at* $t = 0$ *and* $R = 0$*; in general, the absense of a task* $\mathbf{F}$ *implies that* $t_{\mathbf{F}} = 0$ *and* $R_{\mathbf{F}} = 0$*.* **Firsts.top**() *and* **Firsts.secondtop**() *indicate the first stack entry of the top two self-modification sequences.*

the free parameters and to give each task a similar importance. Other weightings may be a source for future work; in this study, the focus is on correcting the influence of the baseline reward velocities of different tasks to avoid catastrophic forgetting, rather than determining the importance of a task.

In addition to Equation 6.1, one may add an additional parameter, which considers a correction for the fact that some tasks have a naturally higher standard deviation around the baseline. If tasks have a low reward variability, this can result in low contributions to $\tilde{V}$ even if this performance improvement might be difficult to achieve and include important modifications. Instead of using the mere difference between the task velocity and baseline to compute task-specific components, $C_i = V_{\mathbf{F}_i} - B_i$, such a correction would compute a Z-score:

$$C_i = w_i \frac{V_{\mathbf{F}_i} - B_i}{S_i}\,, \tag{6.2}$$

for a given task $\mathbf{F}_i$, where S_i is the standard deviation of the task velocity. From here on, the version with $V_{\mathbf{F}_i} - B_i$ will be referred to as the *absolute model*, whilst the version in Equation 6.2 will be called the *relative model*.

Algorithm 6.1 Calculating task-specific reward velocities. When a stack entry e has no time passed for a given task $\mathbf{F}_i$, `getLastEval` gets the top-most entry e' prior to e for which task-time has passed, and uses the task-specific velocity $V_{\mathbf{F}_i}(e')$ as an estimation of $V_{\mathbf{F}_i}(e)$.

```
procedure getTaskVelocity(entry $e$,task $\mathbf{F}_i$, task time $t_{\mathbf{F}_i}$)
    if $e.t_{\mathbf{F}_i} = t_{\mathbf{F}_i}$ then
        $V_{\mathbf{F}_i} \leftarrow$estimateVelocity($e$,$\mathbf{F}_i$,$t_{\mathbf{F}_i}$);
    else
        $V_{\mathbf{F}_i} \leftarrow \frac{R(t_{\mathbf{F}_i}) - e.R_{\mathbf{F}_i}}{t_{\mathbf{F}_i} - e.t_{\mathbf{F}_i}}$;
    end if
end procedure
procedure estimateVelocity(entry $e$,task $\mathbf{F}_i$, task time $t_{\mathbf{F}_i}$)
    $e' \leftarrow$ getLastEval($e.first, F$);
    return $V_{\mathbf{F}_i}(e')$;
end procedure
procedure getLastEval(index $i$,task $\mathbf{F}$)
    $t \leftarrow \mathscr{S}[i].t_{\mathbf{F}}$;
    if $t \leftarrow 0$ then
        return $\emptyset$;
    end if
    $i \leftarrow i - 1$;
    while $i \geq 0$ do
        $first \leftarrow \mathscr{S}[i].first$;
        $e \leftarrow \mathscr{S}[first]$;
        if $e.t_{\mathbf{F}} < t_{\mathbf{F}}$ then
            return $\mathscr{S}[e.first]$;
        end if
        $i \leftarrow i - 1$;
    end while
end procedure
```

6.1.4 The argmax operator: finding the best policy

In traditional SSA, obtaining the best policy is done by comparing the reward velocities of the first entries of the top two self-modification sequences in the stack. In Lifelong SSA there are two complications which make finding the best policy more difficult.

The first and main problem is that the stack is not necessarily sorted. When it is assumed the current task $\mathbf{F}$ has occured in a task block before the current, the current evaluation interval will not affect entries introduced during intermediate tasks because these will have the same task-specific time for $\mathbf{F}$. This implies entries *before* the intermediate task blocks may, due to the influence of the current evaluation interval on the task-specific velocity $V_{\mathbf{F}}$, now have a larger baseline-adjusted global velocity $\tilde{V}$ than these intermediate entries, resulting in an unsorted stack. However, there are two properties which can be exploited to avoid checking all the entries: (a) due to the entries in the bottom of the stack being evaluated over a longer period of time, the current evaluation interval has a progressively reduced effect as the stack is traversed from to bottom; (b) in Equation 6.1, it can be observed that each of the tasks' contributions are independent when the weight parameters are fixed. Lifelong SSA exploits these properties by defining a stop-condition; based on an entry's time of introduction, an upper bound on the change in task-specific velocity [1] allows defining an entry after which, when traversing from top to bottom of the stack, none of the entries could possibly be the new maximum baseline-adjusted global velocity.

A second key problem is that, when parameters such as the task weights, the baselines or the standard deviations change over time, this could change the relative importance of tasks and therefore the maximal velocity of the self-modification sequences may be hidden anywhere in $Firsts \subset \mathscr{S}$. Since these effects do not progressively decline over the stack, all entries must be checked for the new maximum; however, computational expense can easily be limited by limiting the amount of parameter changes. A similar exhaustive check is performed at the start of an unseen task, because for unseen tasks the task-specific timers and cumulative rewards will all be zero; since then all the previous task-specific velocities are the same, there is no progressively declining effect. Due to the need for exhaustive checks, the Lifelong Success Story Algorithm cannot update the parameters w, S and B too often, or it will come at a significant performance cost. For this reason, updates are limited in the experiments to $N_{est} = 10$ estimation intervals of length $T_{est} = 5000$ for each task. To avoid evaluations based on faulty estimates, it is required that at least 3 estimation intervals have ocurred on a new task before any evaluation can take place; these first three intervals follow in quick succession with $T_{est} = 25$.

The resulting algorithm, illustrated in Algorithm 6.2, ensures the top velocity in the stack is maximal, as shown in the proof of Theorem E.1. Note that if it is assumed that each unique task can occur only once, this procedure is not necessary, since then it is sufficient to improve on each task. In terms of time complexity, the worst case is $O(N|\mathscr{S}|)$, where N is the number of tasks seen so far and $|\mathscr{S}|$ is the stacksize, which adds a factor N to the traditional SSA due to the need to combine task-specific velocities, and the best case is $O(1)$ with all entries being accepted due to improving on the top entry. The average case requires more computations than the traditional SSA due to the requirement to go back until the effect diminishes to near zero; however, for longer lifetimes only a small proportion of the stack needs to be checked. Space complexity is the same as in traditional SSA, namely, the best case is when no entry is accepted, leading to $O(1)$ complexity, and the worst case is when all entries are accepted, in which case complexity is $O(NT)$ where T is the lifetime of the learner and N the number of seen tasks, since task-specific timers and cumulative rewards need to be stored. Overall this suggests the algorithm is applicable when the unique number of tasks does not keep

[1]The use of the task-specific velocity relies on the fact that only one task has time passed since last evaluation; this is ensured by forcing evaluation at the end of the task block.

growing and when the valid number of modifications does not grow too large, such as when parameters subject to modification are required to be relatively stable over time.

Algorithm 6.2 An algorithm for evaluating self-modification sequences (SMSs) in Lifelong SSA.

```
procedure popBackUntilNoEffect(current task $\mathbf{F}$, stack indexes $Firsts \subset \mathscr{S}$,
small user-defined tolerance $\epsilon > 0$ )
    set $top \leftarrow Firsts.top()$ and $second \leftarrow Firsts.secondtop()$;
    if Lifelong SSA parameter changed or new task block with $F \in \mathcal{F}$ then
        $maxfirst = $ getMaxVelocity()          ▷ check all velocities and get maximum
        popAndRestore($maxfirst$)               ▷ cf. Algorithm 4.1
        return
    end if
    $\hat{M}_{\mathbf{F}} \leftarrow \max\left(\hat{M}_{\mathbf{F}}, V^{\mathbf{F}}(top)\right)$
    if $V_{\mathbf{F}}(top) \leq V_{\mathbf{F}}(second)$ then
        $max \leftarrow \tilde{V}(second)$
        $maxfirst \leftarrow second$
        $\Delta V^{max} \leftarrow \frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - second.t_{\mathbf{F}}}(\hat{M}_{\mathbf{F}} - V_{\mathbf{F}}(top))$.
        if $\Delta V^{max} \geq \epsilon$ then          ▷ maximum may have changed
            for $e$ in $Firsts.thirdtop() \dots Firsts.bottom()$ do
                if $\tilde{V}(e) \geq max$ then
                    $max \leftarrow \tilde{V}(e)$
                    $maxfirst \leftarrow e$
                end if
                if $V_{\mathbf{F}}(e) > V_{\mathbf{F}}(top)$ then          ▷ This implies $\Delta V_{\mathbf{F}}(e) < 0$.
                    $\Delta V^{max} \leftarrow \frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}}(\hat{M}_{\mathbf{F}} - V_{\mathbf{F}}(top))$.
                    if $\Delta V^{max} < \epsilon$ then:
                        break
                    end if
                end if
            end for
        end if
    else
        $maxfirst \leftarrow top$
    end if
    popUntilMaxfirst($maxfirst$)
end procedure
procedure popUntilMaxfirst(index with maximal global velocity $maxfirst$)
    while True do
        $first \leftarrow Firsts.top().first$
        if $first = maxfirst$ then
            break;
        end if
        popAndRestore($first$);                 ▷ cf. Algorithm 4.1
    end while
end procedure
```

6.1.5 The evaluation-modification gap

In addition to the above, a seemingly minor issue, though with large cumulative effects over time, is observed and corrected for. The issue is a gap in between the time of evaluation and the start of the next self-modification sequence that is unaccounted for by the velocity computations of the next SSA evaluation.

When the current policy is evaluated on the Success Story Criterion at a check-point at time t_1, the traditional Success Story Algorithm does not necessarily start the next self-modification sequence. Instead, it waits for the next self-modification, at some time $t' \geq t_1$, because otherwise the policy at time t_1 is still in place. This has the side-effect that, in the time interval $(t_1, t']$, there is a gap which is not accounted for when comparing only the top two velocities in $Firsts$. This "evaluation-modification-gap" is problematic because when the velocity in between evaluation and modification is much lower than $V(Firsts.top())$, as illustrated in Figure 6.4, the stack may become unsorted. Therefore, checking only that $V(Firsts.top()) > V(Firsts.secondtop())$ is not enough. Note that if the velocity during the evaluation-modification gap is higher than usual, then the top entry will be removed, and there no longer is a gap.

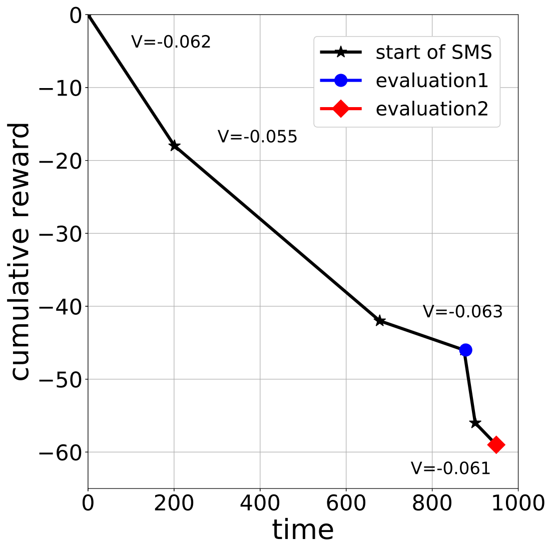


Figure 6.4: *Illustration of the problem with the evaluation-modification-gap. A steep decline in cumulative reward during the gap in between the prior evaluation and the first modification of the final self-modification sequence confuses the SSA algorithm. (a) At the first evaluation, all self-modification sequences form a success story of increasing reward velocity, as observable by the black line's slope. (b) However, at the second evaluation, the velocities indicated by the V-annotations, representing the slopes from the starts to the final evaluation point, show that the stack is no longer sorted: the sequence starting at $t = 0$ as well as the sequence starting at $t = 200$ have a higher reward velocity than the final self-modification sequence starting at time $t = 900$. The effect of pair-wise popping of the stack is that SSA maintains the self-modification sequence starting around $t = 700$ even though the sequence starting at $t = 200$ has the best performance.*

To solve this issue in both traditional and Lifelong SSA, a new self-modification is forced to be executed immediately after each evaluation. This is implemented by using a special instruction `endSelfModAfterNext` which sets a variable $waitForNext \leftarrow True$, to ensure evaluation is stalled until a syntactically correct self-modification is proposed; the evaluation can then proceed after which immediately, with no delay, the proposed self-modification is executed and pushed to the top of the stack.

At the end of a task, Lifelong SSA enforces evaluations to ensure that only a single task has its timer ticking during the current evaluation interval. To ensure there is no evaluation-modification gap in that case, a small probability increase δ, sampled randomly from a uniform distribution, $\delta \sim U(0, 0.001)$, is applied to a randomly chosen cell in the probability matrix.

6.1.6 Metrics to assess Lifelong SSA

In addition to the performance, and the metrics proposed for transfer learning and forgetting ratios mentioned in Section 5.1.6, here two additional metrics are considered to measure the effect of Lifelong SSA when compared to traditional SSA.

Rationale Lifelong SSA is proposed to ameliorate the limitations of traditional SSA when faced with tasks with different reward velocity profiles. The following effects are expected to be observed in traditional SSA and not in Lifelong SSA:

- In the short term, negative transitions result in forgetting: when a task with high reward velocity is followed by a task with low reward velocity, the stack of traditional SSA will reduce in size, whilst Lifelong SSA is unaffected.
- In the short term, positive transitions result in too eager learning: when a task with low reward velocity is followed by a task with high reward velocity, the stack of traditional SSA will grow rapidly in size whilst Lifelong SSA is unaffected.
- In the long term, these two factors make it more difficult to learn how to make self-modifications.

The analysis in Section 6.1.2 and

The effect is assessed with an empirical study of an active adaptive perception system similar to the SMP-DRQN system mentioned in Chapter 4. A theoretical investigation into these effects would

Stack size To assess the long-term effect of Lifelong SSA on learning self-modifications, the metric used here is the stack size, further denoted as $|\mathscr{S}|$. Since the stack $\mathscr{S}$ includes all the valid modifications that the learner made to improve its instruction module or perception module, the number of entries in the stack, or stack size, shows the knowledge accumulated and retained over the lifetime.

Correlation between task velocity and stack size To assess short-term effects, the key thing to show is whether or not positive transitions result in eager learning whilst negative transitions result in forgetting.

To define eager learning and forgetting, the dynamics of the stack size over time are here considered: since the stack size reflects how much knowledge is accumulated, the change in stack size, here denoted as $\Delta\mathscr{S}$, indicates learning, in case $\Delta\mathscr{S} > 0$ and forgetting, in case $\Delta\mathscr{S} < 0$. To define positive and negative transitions, a notion of the reward velocity corresponding to each task is needed. To do so, a natural option, comparable across different learners, is the reward velocity of a policy which outputs random actions, $V_R(\mathbf{F}(t))$ [2], where $\mathbf{F}(t)$ is the task occuring at the t'th task block. A positive and negative transition can then be defined based on the quantity

$$\Delta V_R = V_R(\mathbf{F}(t)) - V_R(\mathbf{F}(t-1))\,, \tag{6.3}$$

which compares the new task's random performance to that of the previous task's random performance. Based on this quantity, a positive transition occurs when $\Delta V_R > 0$ whilst a negative transition occurs when $\Delta V_R < 0$.

To combine both definitions in a single quantity which addresses the key question, a metric is needed which captures the dependency between both quantities. This is here done using the Pearson product-moment correlation coefficient r between the change in task velocity ΔV_R and the resulting change in stack size $\Delta\mathscr{S}$:

$$r(\Delta V_R, \Delta\mathscr{S}) = \frac{Cov(\Delta V_R, \Delta\mathscr{S})}{\sigma_{\Delta V_R}\sigma_{\Delta\mathscr{S}}}\,, \tag{6.4}$$

where Cov computes the covariance whilst σ computes the standard deviation. The interpretation is that if $r > 0$ then eager learning follows positive task transitions and catastrophic forgetting follows negative task transitions.

[2] see also Equation 5.6 and 5.7

6.2 Experimental setup

The purpose of the following experiments is to investigate which evaluation module is better for evaluating incremental self-modifications, with and without the above modifications. The learners incorporated in this study will be simplistic examples of active adaptive perception; their instructions are as simple as possible, while still technically categorising as active adaptive perception learners, affecting perceptual modification and advice only by changing two hyperparameters. This is done to reduce the complexity of the learning systems and thereby facilitate their interpretation.

6.2.1 Experimental plan

All experiments follow the task sequences mentioned in Section 5.2.1. The experiments aim to determine which evaluation module is best suited for the evaluation of active adaptive perception learners in lifelong learning scenarios.

Learning conditions In the experiments, different SSA-based methods have the task to improve a Deep Recurrent Q-Network by setting the exploration rate and the initial learning rate provided to the AdaDelta optimiser [238]. The following learning conditions are included:

- DRQN: as a baseline method, hyperparameters are fixed from start to end of lifetime, without any SSA optimisation.
- SMP-DRQN: a single SSA agent optimises a single set of hyperparameters, and is evaluated based on the lifetime cumulative reward.
- Lifelong SMP-DRQN (Abs): a single SSA agent optimises a single set of hyperparameters, and is evaluated based the adjusted global velocity (cf. Equation 6.1).
- Lifelong SMP-DRQN (Rel): the same as the previous condition, but with the Z-score adjustment mentioned in Equation 6.2).

To isolate the effect of the correction for task velocities, and eliminate any effect of the evaluation-modification gap, all learners will use the `endSelfModAfterNext` instruction. Parameter settings are given in Appendix B, Section B.3.

Implementation details For SMP-DRQN, instructions remain the same compared to the experiments in the previous chapter, except for the following changes: to allow IS to change how DRQN is used and modified, two special instructions `set_eps` and `set_lr`

Table 6.1: *List of instructions used for the instruction set $\mathcal{A}$ in the SMP learners. Instructions are divided in categories based on the module it directly affects: PM for perception module, IM for instruction module, EM for evaluation module, and WM for working memory.* ***Function and operator definitions:*** *c is the working memory tape, often indexed by double/indirect-addressing; $clip(a;[b,c])$ clips a to an integer in the range $[b,c]$. $a//b$ returns $sign(a)*MaxInt$ if $b=0$ and integer division otherwise; $a \bmod b$ returns a if $b=0$ and $a-b*floor(a/b)$ otherwise.* ***Note:*** *some operations yield invalid addresses or numbers according to rules of syntactical correctness (cf. [157]); if these conditions are not met the operation does nothing except for the usual increments to the instruction pointer IP.*

Instruction	Type	Explanation
`getP`	$\mathcal{A}^{IM}$	$c_{c_{a_3}} = round(MaxInt * \mathcal{P}_{c_{a_1},c_{a_2}})$
`incP`(a_1,a_2,a_3)	$\mathcal{A}^{IM}$	push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathcal{S}$. Then, set $\mathcal{P}_{c_{a_1},c_{a_2}} \leftarrow 1-.01*c_{c_{a_3}}*(1-\mathcal{P}_{c_{a_1},c_{a_2}})$, with $c_{a_1} \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$; $\mathcal{P}_{c_{a_1},i} \leftarrow .01*c_{c_{a_3}}*\mathcal{P}$ for all $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$
`decP`(a_1,a_2,a_3)	$\mathcal{A}^{IM}$	push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathcal{S}$. Then, set $\mathcal{P}_{c_{a_1},c_{a_2}} \leftarrow .01*c_{c_{a_3}}*\mathcal{P}_{c_{a_1},c_{a_2}}$, with $c_{a_1} \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$; $\mathcal{P}_{c_{a_1},i} \leftarrow \mathcal{P}_{c_{a_1},i}*(1-.01*c_{c_{a_3}}*\mathcal{P}_{c_{a_1},c_{a_2}})/(1-\mathcal{P}_{c_{a_1},c_{a_2}})$ for all $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0,\dots,\lvert\mathcal{A}\rvert-1\}$.
`set_eps`	$\mathcal{A}^{PM}$	$\epsilon \leftarrow a_\epsilon + a1*(b_\epsilon - a_\epsilon)/(n_{ops}-1)$, with $a_\epsilon = 0.0$ and $b_\epsilon = 0.4$
`set_lr`	$\mathcal{A}^{PM}$	$\alpha \leftarrow a_\alpha + a1*(b_\alpha - a_\alpha)/(n_{ops}-1)$, with $a_\alpha = 10^{-6}$ and $b_\alpha = 0.2$
`endSelfModAfterNext`	$\mathcal{A}^{EM}$	set $waitMod \leftarrow True$; at the first syntactically correct self-modification, perform SSA/Lifelong SSA evaluation; add the self-modification to the stack; set $waitMod \leftarrow False$
`jumpHome`$()$	$\mathcal{A}^{WM}$	set $IP \leftarrow ProgramStart$
`jumpEq`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	if $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP \leftarrow c_{c_{a_3}}$.
`jumpLower`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	if $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP < c_{c_{a_3}}$.
`add`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} + c_{c_{a_2}};[MinInt, MaxInt])$
`sub`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} - c_{c_{a_2}};[MinInt, MaxInt])$
`mult`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} * c_{c_{a_2}};[MinInt, MaxInt])$
`div`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} // c_{c_{a_2}};[MinInt, MaxInt])$
`rem`(a_1,a_2,a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} \bmod c_{c_{a_2}};[MinInt, MaxInt])$
`mov`(a_1,a_2)	$\mathcal{A}^{WM}$	$c_{c_{a_2}} \leftarrow c_{c_{a_1}}$
`init`(a_1)	$\mathcal{A}^{WM}$	$c_{a_2} \leftarrow a_1 - ProgramStart - 2$
`inc`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} + 1;[MinInt, MaxInt])$
`dec`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} - 1;[MinInt, MaxInt])$

mentioned in Table 6.1 choose the learning rate and the exploration rate to be used for DRQN; the `getP` used in some IS implementations is added; the ratio of duplication was similar, but due to a lower number instructions the absolute number was lower, i.e. `incP` and `decP` were represented each 6 times in total; to narrow the focus of the study, `trainReplay` was replaced with a fixed training schedule with update-frequency of once every four time steps.

Another change is that external actions are left entirely up to the DRQN module, removing the need for the `doQuntil` instruction: after DRQN selects an action, Incremental Self-improvement performs exactly 10 cycles at each time step. This is to make computation time more predictable compared to the previous system which allowed a self-chosen number of internal computations before selecting an external action, simplifies analysis, and reduces the search space. Because the only task of the learner is to optimise DRQN's learning, the SSA optimisations are delayed until the DRQN starts learning, which is defined by the replay start time $t = 50000$.

6.3 Results

This section analyses the results concerning the performance and other metrics explained in Section 6.1.6.

6.3.1 Performance comparison

Figure 6.5 illustrates the lifetime average performance of the various learners. Similar to the results in the previous chapter, the performance benefit of active adaptive perception learners can only be observed near the end of the lifetime. In this lifelong learning setting, however, it can be observed that the traditional DRQN's performance is deteriorating rapidly towards the end of the lifetime; presumably this is because its weights are being pulled to opposite directions, never specialising well at any particular task. The active adaptive perception learners seem to be able to sustain a more stable performance level at the end of the lifetime.

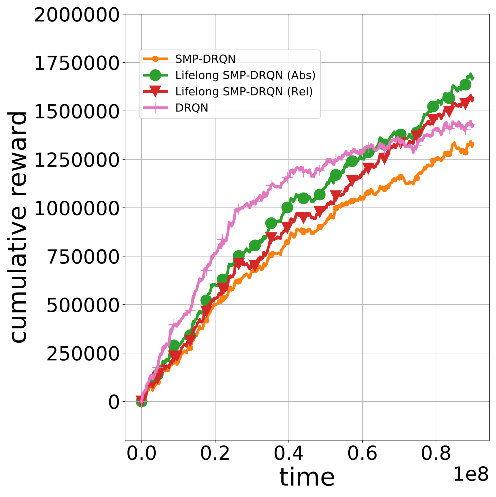


Figure 6.5: *Cumulative reward over time for each evaluation module.*

The variance analysis, shown in Table 6.2 illustrates that the overall performance differences between conditions are quite small compared to the variability within conditions, with no significant effects being found. This suggests that the impact of the condition on performance is small.

The conclusion from the above analysis is not complete, in the sense that one condition may still consistently outperform another condition, even if compared to the variance this effect is small. Therefore, Table 6.3 analyses the results from an ordinal perspective. This analysis shows that, although the rank of performance of different methods is not consistent across the 18 independent runs, the different methods are characterised by a different distribution of performance, as indicated by Cliff's δ. The performance distributions of Lifelong SSA conditions dominate those in DRQN and traditional SSA,

Table 6.2: *Variance analysis on the effect of evaluation module on performance (mean $\pm$ standard deviation). $<$ and $>$ are used to indicate whether the method's performance is higher or lower than its comparison, while p denotes the significance value of the pair-wise F-test. Performance is based on the cumulative reward function as $R(t) = \sum_{\tau=0}^{t} r_\tau$. The* ***lifetime*** *reward velocity is obtained by $R(T)/T$ where $T = 9 * 10^7$ is the total lifetime of the learner, and the* ***final*** *reward velocity is obtained by $\frac{R(T)-R(T-t)}{T-t}$ where $T = 9 * 10^7$ and $t = 2 * 10^6$.*

Metric	**Method**	**Performance**	**Comparison**		
lifetime			Lifelong SMP-DRQN (Abs)	Lifelong SMP-DRQN (Rel)	DRQN
	SMP-DRQN	0.015 ± 0.010	$<$, $p = 0.376$	$<$, $p = 0.565$	$<$, $p = 0.804$
	Lifelong SMP-DRQN (Abs)	0.019 ± 0.007	/	$>$, $p = 0.759$	$>$, $p = 0.444$
	Lifelong SMP-DRQN (Rel)	0.017 ± 0.008	/	/	$>$, $p = 0.690$
	DRQN	0.016 ± 0.007	/	/	/
final			Lifelong SMP-DRQN (Abs)	Lifelong SMP-DRQN (Rel)	DRQN
	SMP-DRQN	0.015 ± 0.058	$>$, $p = 0.784$	$>$, $p = 0.966$	$>$, $p = 0.653$
	Lifelong SMP-DRQN (Abs)	0.008 ± 0.046	/	$<$, $p = 0.819$	$>$, $p = 0.830$
	Lifelong SMP-DRQN (Rel)	0.014 ± 0.057	/	/	$>$, $p = 0.683$
	DRQN	0.002 ± 0.054	/	/	/

Table 6.3: *Ordinal analysis on the effect of evaluation module on performance (median $\pm$ interquartile range). Performance is based on the lifetime reward velocity $R(T)/T$ where $T = 9 * 10^7$ is the total lifetime of the learner.* ***significance*** *denotes the p-value obtained from the Wilcoxon ranksum-test.* ***effect size*** *denotes Cliff's δ, an ordinal effect size metric for dominance of one distribution over the other. The label in parenthesis denotes the magnitude of the effect; its estimate is based on Vargha et al.'s study [221].*

Method	**Performance**	**Comparison**					
		Lifelong SMP-DRQN (Abs)		Lifelong SMP-DRQN (Rel)		DRQN	
		significance	effect size	significance	effect size	significance	effect size
SMP-DRQN	0.015 ± 0.009	$p = 0.121$	-0.30 (medium)	$p = 0.486$	-0.14 (small)	$p = 0.635$	-0.09 (negligible)
Lifelong SMP-DRQN (Abs)	0.018 ± 0.007	/	/	$p = 0.752$	0.06 (negligible)	$p = 0.282$	0.21 (small)
Lifelong SMP-DRQN (Rel)	0.018 ± 0.011	/	/	/	/	$p = 0.681$	0.08 (negligible)
DRQN	0.015 ± 0.007	/	/	/	/	/	/

with non-negligible effect sizes. The largest effect here is the medium effect size of $\delta = -0.30$ when SMP-DRQN is compared to Lifelong SMP-DRQN (Abs).

Task performances are overall quite similar, although DRQN and SMP-DRQN with traditional SSA have a degrading performance effect over time, as illustrated in Figure E.1 in Appendix D.

6.3.2 Effect of task velocities on stack size

Here the metrics to assess long- and short-term effects on the stack size, mentioned in Section 6.1.6, are analysed. Results, illustrated in Table 6.4, are in line with the hypotheses mentioned in Section 6.1.6:

- An improved ability to build and maintain knowledge on a long-term basis: the stack size of the Lifelong SSA learners is on average 70-90 times larger than that of the traditional SSA learner.

Table 6.4: *Effect of task velocities on stack size. $r(\Delta V_R, \Delta S)$ is the Pearson product-moment correlation between the change in the random baseline velocity, ΔV_R, and the corresponding average stack size change $\Delta\mathscr{S}$. p denotes the significance test for one-tailed t-test. $|\mathscr{S}|$ denotes the stack size obtained at the end of the lifetime, for which both the average and standard deviation across runs is reported.* ***Abs*** *denotes the absolute model according to Equation 6.1 while* ***Rel*** *denotes the relative model according to Equation 6.2.*

Method	**Evaluation type**	$r(\Delta V_R, \Delta\mathscr{S})$	$\lvert\mathscr{S}\rvert$
SMP-DRQN	Traditional SSA	0.633, $\mathbf{p < 0.001}$	20 ± 14
Lifelong SMP-DRQN (Abs)	Lifelong SSA (Abs)	-0.189, $p = 1.000$	1405 ± 1099
Lifelong SMP-DRQN (Rel)	Lifelong SSA (Rel)	-0.169, $p = 0.999$	1810 ± 868

Note: Task transitions are recorded across the entire lifetime of the learner. To calculate $\Delta\mathscr{S}$, the stack sizes are compared the end of the previous task, at time t, and then after the initial phase of the new task, at time $t' = t + 10000$.

- No eager learning and catastrophic forgetting: for Lifelong SSA no positive correlation is observed between task velocity and stack size, whilst for traditional SSA this is the case.

6.3.3 Effect of prior tasks on forgetting and transfer

A key reason why lifelong learning is so difficult is because, when learning a new skill or maintain an old skill, many different tasks are presented and this can either improve or diminish performance. Figure 6.6 shows these two effects graphically, using the forgetting ratio and transfer ratio mentioned in Section 5.1.6. In these plots, no clear difference can be observed between the traditional SSA and Lifelong SSA. Given the above-mentioned results on performance and other metrics, it is clear that there is a benefit from Lifelong SSA; the absense of an effect in these plots may therefore mean that the effects of adding or removing incremental self-modifications made by the instruction module do not come into effect immediately, but rather have effects only over the long-term and therefore calculating the effect across the next task block has little effect. Comparisons to DRQN, however, show that transfer learning is reduced whilst forgetting is improved. This interaction effect may be because a reduced forgetting on some tasks implies a worse performance on other tasks due to negative transfer.

6.3.4 Development of the exploration rate and learning rate

Since in this study, the difference between the learners is how they modify the exploration rate and the learning rate, the performance effect is due to the dynamics of these hyperparameters. Figure 6.8 shows that, after an initial phase of searching across the range of the hyperparameters, SMP-DRQN implementations choose a higher exploration rate and learning rate than DRQN, with an average of around 0.25 for the exploration rate and 0.13 for the learning rate. The SMP-DRQN based on traditional SSA has a

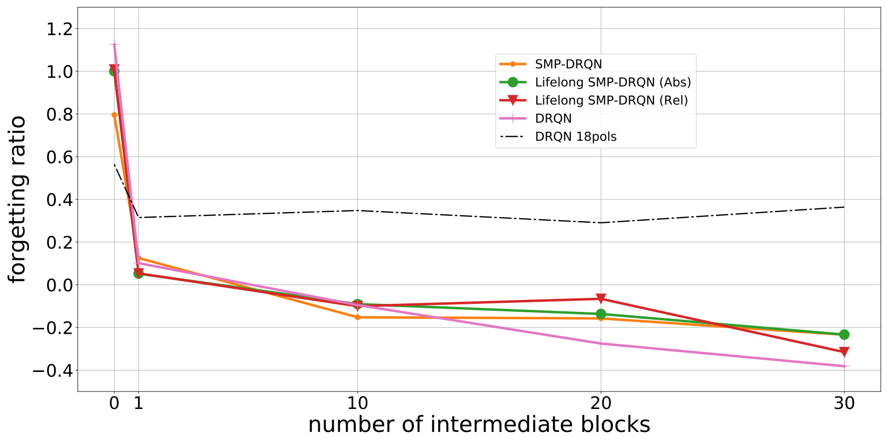


Figure 6.6: *Effect of intermediate tasks on forgetting for different evaluation modules. The x-axis shows the number of interfering tasks, measured in bins; for example, $x = 1$ means a number of interfering tasks in $[1, 9]$ and the final value of $x = 30$ means 30 or greater. The y-axis shows the forgetting ratio as defined in Equation 5.6, with the interpretation being that, $y = -1$ means a decrease of 1 time the random performance, whilst $y = 1$ means an increase of 1 time the random performance, when compared to the performance of the same task in a task block before the interfering task blocks.* ***Note:*** *the 18-policy condition (dashed line) can be seen as a baseline since there is no catastrophic forgetting possible because each of its policies sees only a single task; therefore every learner below the 18-policy will have some forgetting.*

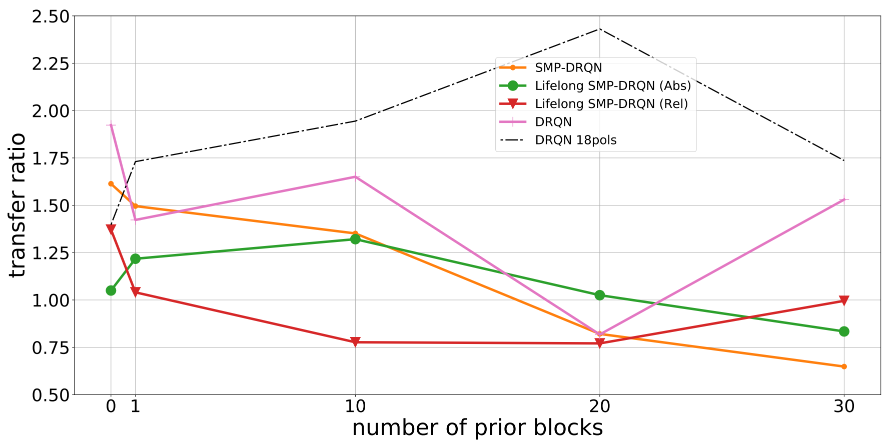
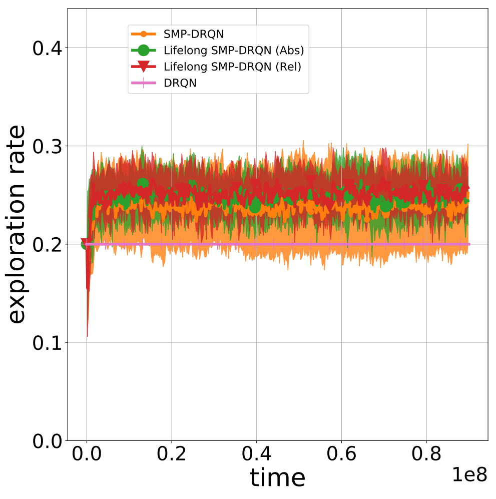
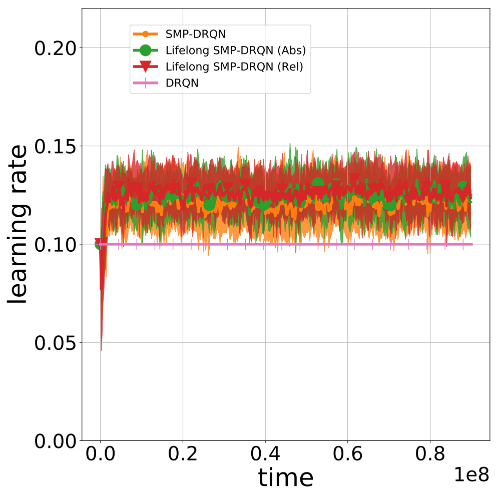


Figure 6.7: *Effect of prior tasks on transfer for different evaluation modules. The x-axis shows the number of prior tasks, measured in bins; for example, $x = 1$ means a number of interfering tasks in $[1, 9]$ and the final value of $x = 30$ means 30 or greater. The y-axis shows the forgetting ratio as defined in Equation 5.7, with the interpretation being that, as a result of the prior task blocks, $y = -1$ means a decrease of 1 time the random performance, whilst $y = 1$ means an increase of 1 time the random performance, when compared to the random performance.* ***Note:*** *the 18-policy condition (dashed line) can be seen as a baseline since there is no transfer possible because each of its policies sees only a single task; therefore every learner below the 18-policy will have negative transfer.*



(a) Exploration rate



(b) Learning rate

Figure 6.8: *Development over hyperparameters over the lifetime. Central lines represent the mean value of the hyperparameter across independent runs, whilst the shaded area represents the variability based on the standard deviation.*

larger variability than the methods based on Lifelong SSA. This is because the former maintains only a limited number of self-modifications in the stack, as shown in Table 6.4; this implies that the instruction module's policy parameters are closer to the initial $\mathcal{P}^0$, which applied a uniform distribution in $[0, 0.40]$ for the exploration rate and in $[10^{-6}, 0.20]$ for the learning rate. With the performance of the DRQN system declining over time, the interpretation of these findings is that maintaining a large learning rate is necessary to maintain plasticity in the neural network, to learn new behaviours but especially to re-learn forgotten behaviours, and that maintaining a large exploration rate is necessary to supply the right data.

6.3.5 Resource consumption

The time complexity of Lifelong SSA does not appear to be a bottle-neck in practice. The total run time of traditional SSA is similar to the total run time of Lifelong SSA, both being around 12 days for all 90 million time steps to finish; all runs are done on a single Intel Xeon E5-2670 CPU (2.60GHz) on the IRIDIS4 supercomputer [215]. The effect of the 10 additional IS cycles each time step can be estimated by the observation that DRQN needed around 9 days for all 90 million time steps. The time estimation concludes that the bulk of the time is being consumed by the DRQN's forward and backward passes, and that approximately 30% of computation time is added by performing IS cycles, with a negligible amount of computation time increase due to the Lifelong SSA evaluation compared to SSA evaluation. The observed memory consumption by the stack ranges between 1-2MB for Lifelong SSA and between 40-120 kB in traditional SSA.

6.4 Discussion

6.4.1 Strengths

This chapter demonstrates that a utility function which accounts for different reward velocities observed in different tasks ameliorates the catastrophic forgetting exhibited by the traditional Success Story Algorithm. Experiments on challenging task sequences in partially observable environments demonstrate that this utility function improves the performance of the SMP-DRQN implementation by removing the correlation between tasks' baseline reward velocities and the acceptance of new self-modifications. Due to correcting for tasks' baseline reward velocities, the proposed utility function is expected to yield performance benefits when different tasks are widely differring in their reward function, although there is no reason to believe it would be detrimental to performance when this is not the case.

A novel evaluation instruction is also proposed to ensure no gap between the evaluation and the first modification of a self-modification sequence can exist. Although a fairly minor change and none of the learners included in the ablation study excluded this modification, its effect could potentially be large as the original instruction does not guarantee that all self-modifications are progressively improving reward intake.

To allow the evaluation of self-modifications with the baseline-adjusted global velocity as a utility function, the SSA algorithm is modified, resulting into Lifelong SSA. The advantages of SSA also hold true for the novel Lifelong SSA: its long-term evaluation makes it possible to learn in non-episodic environments, environments with no artificial time-outs, and sparse-reward environments; SSA-type algorithms are useful for meta-learning in long-term environments due to the ability to track back to earlier versions of a set of parameters across the entire lifetime. Similar to SSA, the use of Lifelong SSA is not limited to learners based on Incremental Self-improvement or learners with active adaptive perception. For example, the Adaptive Levin Search [166] approach could use Lifelong SSA instead of SSA for improved lifelong learning. The contribution may possibly go beyond SSA, since there may similarly be benefits to using the baseline-adjusted global velocity as a utility function in other learning algorithms.

6.4.2 Limitations

There are some points relating to SSA that are unaddressed by the Lifelong SSA evaluation. The Success Story Algorithm consumes a lot of memory when lifetime is long, assuming the algorithm keeps finding self-modifications that improve the reward intake. In the worst case, the memory could grow with one stack entry per time step, if the number of self-modifications is limited to at most one per time step, as in the experiments. In the best case, however, the memory required is only the initial stack-entry.

Other memory aspects remain constant across the lifetime. If the time metric is the number of external actions, then the time consumption could grow in principle indefinitely as the system's reward velocity does not need to take into account real-time cost, however, simply adjusting the time metric to take into account real-time removes this effect. Further, there are no known guarantees about sample complexity, meaning that it cannot be known in advance how much experience is needed to obtain a desirable performance level; this is especially so because the learning itself depends on instructions being computed in a stochastic manner.

Compared to SSA there are three additional limitations. First, Lifelong SSA requires the extra assumption that the task needs to be uniquely and correctly identifiable by the learner. The requirement of task information is an important limitation of Lifelong SSA; however, now that this thesis has helped identify the impact of a tasks's reward structure on the formation of valid self-modifications, it may be relatively straightforward to automatically classify tasks such that the algorithm does not requires knowledge of the current task, based on an automated task identification methods (e.g., [121]). Second, due to the need to correct for task velocities an additional multiplicative factor is added to the time and memory consumption: because all N seen task's reward velocities must be calculated, the baseline-corrected global velocity is a factor N more expensive than the traditional SSA's reward velocity; similarly, the stack structure needs to store timers and cumulative rewards for each of the N seen tasks. Third, the algorithm has a worst average case complexity compared to traditional SSA as it often requires more stack entries to be checked to ensure the top entry is maximal; the extent of this difference will depend on the domain being solved. Consequently, the algorithm in its current form is practically applicable only when the unique number of tasks does not keep growing and when the valid number of modifications does not grow too large.

Although in the present experiments a Lifelong SSA does not demonstrate any observable computation time increase compared to traditional SSA, and its observed memory consumption is at most 2MB, the additional complexity introduced by Lifelong SSA could become problematic once lifetimes grow larger. For scalability therefore, alternatives must be considered, meaning the present study resides at the proof-of-concept level rather than as an implementation recommended for users. Still, it is worth suggesting some implementation tweaks which may remove this limitation: (a) using a real-time version of Lifelong SSA, which conform to earlier SSA experiments uses the real-time in the calculation of the reward velocity to automatically penalise time consumption from the evaluation procedure; (b) adding a penalty to the reward velocity calculation when memory grows too large; (c) using compression techniques to reduce the memory consumed by the stack; and, (d) making algorithmic improvements to the popping algorithm of Lifelong SSA or providing tighter bounds on the maximal velocity change size.

6.5 Summary

In long-term unknown environments, different tasks may be presented in sequence, requiring lifelong learning. A key requirement for lifelong reinforcement learners is a suitable utility function which defines their objective. While active adaptive perception aims to improve the generality of reinforcement learners across long time-scales, the implementations presented in Chapter 4 are based on the Success Story Algorithm (SSA) which has not been tested on challenging lifelong learning scenarios. This chapter analyses the limitations of SSA in such scenarios, particularly in regard to catastrophic forgetting, and based on this analysis proposes a novel long-term utility function which corrects for the different reward functions associated with different tasks. To allow SSA to learn with a different utility function, its algorithm must be modified and this results in Lifelong SSA. SSA and Lifelong SSA are compared empirically on challenging 18-task lifelong learning scenarios, with the main task of SSA learners being to optimise a perception module's hyperparameters. Results show that, although performances are close, correcting for the task-velocities improves the Success Story Algorithm's lifetime performance by removing the positive correlation between acceptance of new self-modifications and a task's baseline performance, and thereby increasing overall stack sizes, reflecting a higher quantity of valid self-modifications. The key contribution of this chapter is a better understanding of the impact of long-term utility functions in lifelong learning environments.

Chapter 7

Discussion

This chapter reflects on how the thesis adresses the research questions, discusses the limitations of the thesis, and suggests future work.

7.1 Strengths

The thesis investigates the role of adaptivity to deal with long-term unknown environments. This includes how adaptivity may benefit in such environments, the potential trade-offs to consider when applying adaptive learning systems, as well as the challenges related to prolonged tasks with sequential dependencies and a multitude of tasks presented in sequence.

7.1.1 Benefits from increased adaptivity

Increased adaptivity is suggested in the thesis as a means to overcome the many unknowns when facing long-term reinforcement learning environments with limited prior knowledge.

The studies presented in the thesis investigate adaptivity to overcome three types of unknowns.

A first unknown is which learning algorithm is suitable for the environment, and active adaptive perception is here put forward as a method to solve this unknown. Active adaptive perception is shown in Chapter 4 to learn how to learn, as evidenced by the increase the rate of adding valid modifications to the perception module and the instruction module. The implementations with the NEAT-based representations are shown to increasingly learn which weight updates may work for a simple feedforward network; and, the SMP-DRQN implementation is shown to increasingly learn which experiences are useful to serve as goal, and when to use which instructions. By contrast, the SMP alone, based on Incremental Self-improvement [157], could not obtain a high performance, and the results indicate that perceptual modification and usage instructions are the most impactful on performance. Active adaptive perception therefore presents a departure from existing self-modifying policies. Previous theoretical approaches to self-modifying policies [161] have yet to be implemented. Practical approaches have either used unadaptive, unrealistic, unscalable, and/or inefficient active perception instructions [165, 164], or, as in the baseline SMP included in this study, rely solely on working memory for processing sensory inputs [157]. Although more work is needed to demonstrate the potential of the generic architecture, Chapter 4 demonstrates the feasibility of learning how to apply, train, construct and provide data to a neural network. Compared to previous meta-learning research in deep reinforcement learning [235, 198], such systems include adaptivity across the entire learning cycle rather than isolating just one of the components, and rather than optimising a single meta-level [40] they optimise the algorithm on an unlimited number of meta-levels. In comparison, the use of self-modifications, to learn which instructions to use, and the use of arguments to provide additional flexibility.

Another unknown investigated in the thesis is the best choice of the hyperparameters. In Chapter 6 adapting the learning rate as well as exploration rate provides a minor source of performance improvement by optimising it based on the lifetime reward acceleration. Chapter 4 demonstrates the potential benefits of goal-based exploration: difficult or uncertain areas are in need for more aggressive exploration. Both findings support one of the principles of active adaptive perception: learning parameters need not be static entities but may be optimised within the agent's current context to facilitate improved learning. Most hyperparameter tuning strategies [50] ignore this potential and require multiple independent runs which may not be possible when online adaptation is required. There have been some works which have included adaptivity to the hyperparameter. Meta-policy gradient [235] temporarily adapts hyperparameters by following a gradient based on a short-term reward contribution, which may not be suitable in many long-term environments; that said, gradient-based optimisation is efficient when the cost function is well-formulated and when the global optimum can be reached. Hyperparameter Optimisation on the Fly [132] can learn dynamic hyper-parameter schedules but only for parameters which are directly related to the policy update function; this includes the learning rate, the discount factor and the parameter for Generalised Advantage Estimation. The performance benefits observed in the thesis also illustrate that adaptive gradient methods such as the AdaDelta [238], included in Deep Recurrent Q-Networks and in the perception module of SMP-DRQN, can be subject to further improvements.

A third unknown investigated in the thesis is which representation may best be chosen for guiding external action selection. As shown in Chapter 4, the active adaptive perception systems can choose when to use which representation, depending on long-term experience or short-term perceptual clues from the environment. Although hierarchical approaches [201, 26, 207] can also adaptively learn which representation to select, the pre-training phase implies the representations themselves are no longer modifiable once they are put in practice. This may be desirable in some cases to reduce the sample complexity; however, for long-term environments an increased adaptivity may in principle yield a better final performance, as suggested by the SMP-DRQN versus DRQN comparisons in both Chapter 4 and 6. The study in Chapter 5 demonstrates that specialising action selection representations to a cluster of tasks is beneficial to performance, thereby presenting an alternative approach to hierarchical lifelong reinforcement learning.

Although adaptivity has also improved performance in the multiple policy study of Chapter 5, the active adaptive perception provides a more novel perspective with comparatively more adaptive potential to explore. In what may be best phrased as *algorithmic emergence*, new learning algorithms and solving strategies may be discovered as a result of the processes emerging from active adaptive perception systems. Although in principle any self-modifying policy may be able to lead to algorithmic emergence, this thesis shows that the use and modification of the perception module is key to unleashing this potential. The work in Chapter 4 effectively demonstrates the emergence of active exploration algorithms [12, 117], where the approach is to increase exploration when learner is uncertain what to do. Similarly, the effects of the increased network usage could also be compared to annealing schedules of the exploration rate in off-policy reinforcement learning methods. The fact that similar behaviours are often hand-crafted by human experts indicates that algorithmic emergence is feasible within this framework.

7.1.2 Insights into the trade-off between adaptivity and efficiency

Since adaptivity depends on the ability to modify free parameters, adaptivity is tightly correlated with the capacity, as it has been well-documented in theoretical work on neural networks that free parameters increase the capacity of learners, allowing them to represent more complex functions or remember more patterns [1] [13, 59, 53]. However, there is a trade-off: when learners with lower capacity and few free parameters can still represent the functions of interest, they will learn more rapidly because they have a smaller parameter space to explore; in other words, they have a higher expressive efficiency [183, 36].

Chapter 4 and 6 demonstrates this trade-off between the adaptivity and the number of samples required for learning. There it is demonstrated that, when comparing SMP-DRQN to DRQN, active adaptive perception learns slow initially but then eventually

[1]As some earlier works suggest [59], storage capacity and representation capacity are intimately intertwined, with the ability to represent functions being equivalent to the ability to memorise a function of the data.

reaches a greater performance. Since the learning itself is also parametrised, the SMP-DRQN system has more free parameters than DRQN. Consequently, it may take more time to learn but, when given sufficient amount of learning time, it can exploit these additional flexibilities to reach an elevated final performance.

Chapter 5 demonstrates that improved capacity in the form of multiple policies, each specialised on a cluster of tasks, is beneficial for learning multiple tasks in sequence, but requires more memory. The results from the task capacity analysis further appear to support the existence of trade-off between task capacity and task-specific performance. DRQN can learn at most 2 tasks, while PPO can learn all 18 tasks included in the analysis, but the 18-policy DRQN far outperforms the 18-policy PPO. Based on transfer and forgetting metrics, the suggested interpretation is that PPO is based on a conservative clipped objective that prevents too large parameter updates while DRQN performs more aggressive updates resulting in erasing previously learned knowledge. This suggests a wider principle in which short-term adaptivity may lead to a superior single-task performance, but, due to overfitting on short-term data, an inferior multi-task performance. This trade-off indicates two feasible approaches to lifelong learning: one in which a learner is organised modularly with specialised representations, and another in which a single representation is learned slowly to ensure the data from all tasks is learned.

Chapter 5 shows that including the additional ability to adapt which policy to use for which tasks in some cases helps efficiency whilst in others it hampers efficiency. When the policies are sufficiently diverse, as is observed for DRQN, learning by switching the policy is more rapid than traversing the parameter space with slow gradient-based updates. However, when the policies are all similar to each other, as is observed for PPO, there is no advantage to be gained from switching, as simply optimising one policy would start from the same region of parameter space and find the best local solution there. This finding corroborates results from various other studies. For example, snapshot ensembling [170, 80] maintains several local minima solutions of a neural network to construct a higher-performing neural network ensemble. Another example is the Intelligent Trial and Error algorithm [39] which enables fault-recovery by searching across a behaviourally diverse repertoire of controllers.

7.1.3 Overcoming challenges in long-term environments

Reinforcement learning in long-term environments is challenging, not only when one task is presented during an extended amount of time, but also when multiple tasks are presented in sequence. The thesis demonstrates insights to overcome these challenges, including the choice of the objective function as well as how to avoid catastrophic forgetting and stimulative positive transfer.

Challenges in single-task environments When the learner has to solve a task for an extended amount of time, there are certain sequential dependencies introduced by the interactive and non-episodic learning setting. Choices made early in the lifetime affect how well the learning proceeds later in the lifetime.

Certain states in the environment may only be reachable by performing particular sequences of behaviours, and this problem is exacerbated when there is no time-out and only sparse rewards. Chapter 4 illustrates what can go wrong if a learner cannot escape a certain region of the state space: the DRQN agent gets stuck in detracting rooms and corridors, preventing it to obtain any rewards at all. Selective network usage and targeted exploration are able to overcome this issue, owing in part to the Success Story Algorithm's long-term objective which can detect stagnation or deterioration over long time intervals. This strategy may therefore provide an alternative to intrinsic rewards [27, 17, 28, 158], which may optimise the intrinsic rewards rather than external rewards and go for short-term rather than long-term gains, or other strategies to overcome sparse reward structures [44], which assume the environment is a controllable simulation that requires time-outs and different learning phases.

Research into environments without time-outs is scarce. One other approach, applied to a dynamic maze setting without time-outs, is to consider sequential dependencies as sub-tasks and define a Q-table for each sub-task [89]. This is reminiscent of hierarchical approaches, and one possibility is to consider automated procedures to learn skills [209, 26]. Compared to these approaches the active adaptive perception systems do not require pre-training or domain knowledge, giving them an advantage when there is no possibility for extensive experimentation before application, as is the case in, for example, planetary exploration missions.

The active adaptive perception learners are likely scalable to more complex sequential dependencies than those investigated in the thesis. Prior research into Incremental Self-improvement [165, 241] has shown the feasibility to learn more complex sequential dependencies involving consecutive steps of cooperation between multiple agents. With the results demonstrating all active adaptive perception implementations to have large performance benefits over the traditional Incremental Self-improvement, it therefore appears plausible that active adaptive perception learners will be able to solve such scenarios with higher performance levels.

Challenges in multi-task cases A second challenge is that a multitude of tasks, presented sequentially across the lifetime, is difficult to learn due to the need to avoid catastrophic forgetting, to selectively transfer and apply knowledge, and to maintain high efficiency and low memory consumption.

Scalability is one important challenge to be overcome for lifelong reinforcement learners. To assess the limit to the number of tasks a single policy can learn and represent without erasing earlier tasks, the thesis shows that the use of multiple policies can provide the basis for (a) an empirical study into the task capacity of learners; and, (b) an analysis of both the rate of learning as well as forgetting and transfer. Based on this analysis, Chapter 5 demonstrates that PPO has a larger task capacity than DRQN. As an explanation for this finding, DRQN has strong single-task performance but suffers from catastrophic forgetting on multiple tasks, while a transfer between a larger number of tasks is possible for PPO. These findings are in line with the literature, in the sense that: (a) PPO transfers well across tasks [130, 28]; and, (b) DQN [126], the predecessor of DRQN, often performs better when learning task-specific policies, as opposed to attempting to overcome catastrophic forgetting with a single policy [95]. Although there have been studies in assessing capacity [13, 59, 53, 220], these have focused on specific types of architectures and are not addressing lifelong reinforcement learning and its practical difficulties arising from the learning algorithm and the data provided. The task capacity analysis therefore represents a practical method to assess scalability in lifelong reinforcement learning.

Although here mainly proposed as a tool for comparing different base-learners, the use of multiple policies each applied onto their own specialisation of tasks may represent a viable approach to overcome the challenges in lifelong reinforcement learning. In comparison to policy reuse [142, 49], learning with multiple policies can be seen as an online, model-agnostic method with a fixed capacity in terms of the number of stored policies, without the need for temporary policies for new tasks. Compared to other hierarchical approaches [209, 26] it may be beneficial when tasks come in similar clusters. Compared to ensemble methods [37], it may represent a computationally cheap alternative, particularly when the number of tasks is limited or the number of policies required is low. Also, in comparison to single-policy approaches, there may be benefits: there is no need to know the task feature vector [151, 42], and there is no need to find a common representation for widely differing tasks [95, 110].

The thesis provides novel insights into the choice of the objective function for learning a complex task sequence. As shown in Chapter 6, the variety of reward functions in the environment poses a significant challenge to learners based on the Success Story Algorithm. To overcome this issue, the thesis demonstrates that correcting for the tasks' different reward profiles allows to maintain a larger amount of knowledge in the stack which records the successful self-modifications across the lifetime. Although the performance effects there are small, more impactful types of self-modifications which go beyond adapting hyperparameters may yield more impressive performance benefits. Also, this objective may be beneficial to other types of reinforcement learners, and a similar objective may also be applicable to supervised learning. Further, in Chapter 5, the larger task capacity of PPO compared to DRQN, and the comparison of the rate of learning, suggests that more conservative objectives may be needed when solving multiple tasks with a limited number of policies across a vast lifetime.

7.2 Limitations

The present study is limited in its theoretical analysis. Active adaptive perception systems such as those in Chapter 4 and 6 are opaque and difficult to analyse theoretically because even their learning algorithm is being modified continually. Similarly, the task capacity analysis in Chapter 5 is purely empirical and the domain to which its conclusions may be generalised is not clear. Since the environments of application may be unknown, the selection of tasks may not be representative. Also, it is likely that, as the task space becomes exhaustively covered with some precision, any new unique tasks will only deviate in a minor sense from learned tasks; consequently, when scaled towards lengthy lifetimes, there may be little or no benefit to learning new policies from scratch because some of the policies are already well-suited to solve such tasks [2].

This thesis only has explored a limited domain in the space of environments. Although the case studies include features such as partial observability, lack of time-outs, and different sequentially presented tasks, there are other features not considered in the thesis which may be of similar importance to reinforcement learning in long-term unknown environments. After longer operation times, memory becomes a bottle-neck which may be problematic for the methods considered in this thesis. Some of the agent's sensors or actuators may be disrupted or fail, either persistently due to damage or sporadically due to temporary environmental influences. Decision delays due to long-lasting computations may affect performance in real-time environments. In similar spirit, some environments may require immediate adaptation and do not allow slow reinforcement learning. The state space and/or state transition dynamics are relatively simple compared to typical deep reinforcement learning applications. Similarly, within the features investigated, the number of included benchmarks is limited. Finally, the current experiments are based on simulation, while the assumption is that the agent is learning online in an unknown target environment which often may be a physical application.

7.3 Future work

Before being applied widely in long-term unknown environments, reinforcement learners must be subjected to more exhaustive testing procedures. Beyond corroborating the present results with similar empirical studies, this includes testing: (i) their ability to scale towards much longer time scales than those in the thesis; (ii) their robustness to minor changes in the environments; (iii) their generalisation to unseen tasks; (iv) their ability to scale to more complex state transition dynamics; (v) application in physical environments.

[2] Although exhaustion of the task space would be a concern for the current formulation of the task capacity analysis, this would also have beneficial implications for the scalability of learning with multiple policies as a lifelong learning method.

A significant driver of the future development of reinforcement learners for long-term unknown environments should be inspired by its possible applications. Long-term autonomous robotic systems is a good example which incorporates these demands, with the limited observations, sparse rewards, and the possibility of landing in a completely unknown environment for time periods much longer than any previous reinforcement learning experiment, and the continuity of the environment where each cognitive or external action co-determines the future. In this context, robots may be sent on isolated missions in space or the ocean bed for topographical or chemical investigations or to continually inspect and maintain the functionality of equipment. Another example application is virtual reality environments, where a virtual agent possibly interacts with human online players, which can often result in sparse, unstable, noisy, or otherwise deceptive reward profiles.

However, another driver for future reinforcement learning systems should be based purely on their algorithmic properties. Further work in algorithmic emergence is needed to answer two questions: (i) under which conditions can meta-reinforcement learning systems construct an existing (near-)optimal algorithm?; (ii) what types of *new* algorithms can emerge from meta-reinforcement learning systems, leading to novel solutions not yet conceived by humans?

The current lifelong reinforcement learning approach may be extended further. First, because the aim has been to show in the ideal case how much could be gained from the proposed lifelong learning strategies, the task index has been given to the learner in the present studies, and the ability of the proposed methods to be integrated with task identification methods should be further investigated. Second, there is a need to further investigate the potential of utility functions which compute the lifetime reward intake with corrections for the different task's reward functions; this includes the study of different weighting schedules to best reflect the expected importance of different tasks, but also investigating the potential of the baseline-adjusted global velocity or similar utility functions to be applied to other reinforcement learning algorithms than SSA, or even applying a similar objective in supervised learning contexts.

More theoretical analysis is required to understand reinforcement learning systems in long-term unknown environments. Although it may be difficult to make general statements about sample complexity of Incremental Self-improvement-based learners, due to the user-defined instructions varying and due the self-modifying learning, future research could investigate this for some simplified cases using a Probably Approximately Correct analysis [217, 70]; these investigations should ideally be coupled to above-mentioned issues in algorithmic emergence. To ensure the task capacity analysis can be tailored to the application, there is a need for theoretical framework to determine key variables for the experimental set-up used to estimate the task capacity: the length of a task block, the number of unique tasks, the type of tasks based on a formal specification of the domain, etc. Further, there is a need for an asymptotic analysis tool to investigate the task capacity as a function of increasing number of tasks and increasing lifetime of the agent.

More work is needed to enable more efficient learning with a given representational capacity. One example that comes to mind is that providing the active adaptive perception learners with an initial bias could prove worthwile to reduce the initial learning time. Moreover, although in this work it is shown that the final performance of the active adaptive perception learners is improved after lengthy lifetimes, more work needs to be done in establishing the conditions in which this improvement will manifest. Further work should also be done to improve the memory requirements and other scalability issues of both active adaptive perception systems and multiple policy systems.

Chapter 8

Conclusion

Increasingly, there is a demand for autonomous artificial agents which are able to learn in a long-term environment with only limited prior knowledge and no ability to fine-tune the system to the environment. The thesis investigates the challenges in long-term unknown environments: limited observations, sparse feedback, long-term dependencies without any intervention from the designer, and multiple reward functions and dynamics characterising different tasks. To improve reinforcement learning in such cases, the thesis investigates long-term adaptivity.

In a first case study, the thesis investigates the challenge of an unknown task occurring for a prolonged amount of time without episodic boundaries and with the possibility of getting no feedback at all. In this scenario, the thesis demonstrates a novel meta-learning principle called active adaptive perception. The principle involves learning how to modify and use perception, and combines strengths of universalist and sub-symbolic AGI to maximise lifetime reward intake. The results show how this system autonomously construct algorithms for learning and exploring its environment and demonstrates selective use of the perception module. The results further show a performance benefit compared to Deep Recurrent Q-Networks, a state-of-the-art deep reinforcement learner for partially observable environments.

In a second case study, the thesis investigates the challenge of multiple tasks presented in sequence. The study demonstrates the effect of learning with multiple policies applied selectively on a subset of the tasks. First, Deep Recurrent Q-Networks learns rapidly and its performance is strongly increased by including multiple policies, whilst Proximal Policy Optimisation learns slowly, allowing improved long-term adaptation to different tasks, and is therefore only weakly affected by including multiple policies. Second, adaptively changing the assigment of policies to tasks can be beneficial if there is sufficient diversity across the policies. A further study in lifelong learning shows that lifetime cumulative reward intake is a misleading objective; instead, the learner must maximise an objective which corrects for the different reward functions of the different tasks.

Appendix A

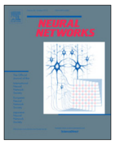
Publications

Neural Networks 115 (2019) 30–49

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet



Learning to learn with active adaptive perception

D.M. Bossens [*], N.C. Townsend, A.J. Sobey

Building 176 Boldrewood Innovation Campus, University of Southampton, Burgess Road, Southampton SO16 7QF, United Kingdom

ARTICLE INFO

Article history:
Received 23 March 2018
Received in revised form 6 March 2019
Accepted 12 March 2019
Available online 25 March 2019

Keywords:
Adaptive perception
Inductive bias
Self-modifying policies
Reinforcement learning
Partial observability

ABSTRACT

Increasingly, autonomous agents will be required to operate on long-term missions. This will create a demand for general intelligence because feedback from a human operator may be sparse and delayed, and because not all behaviours can be prescribed. Deep neural networks and reinforcement learning methods can be applied in such environments but their fixed updating routines imply an inductive bias in learning spatio-temporal patterns, meaning some environments will be unsolvable. To address this problem, this paper proposes active adaptive perception, the ability of an architecture to learn when and how to modify and selectively utilise its perception module. To achieve this, a generic architecture based on a self-modifying policy (SMP) is proposed, and implemented using Incremental Self-improvement with the Success Story Algorithm. The architecture contrasts to deep reinforcement learning systems which follow fixed training strategies and earlier SMP studies which for perception relied either entirely on the working memory or on untrainable active perception instructions. One computationally cheap and one more expensive implementation are presented and compared to DRQN, an off-policy deep reinforcement learner using experience replay and Incremental Self-improvement, an SMP, on various non-episodic partially observable mazes. The results show that the simple instruction set leads to emergent strategies to avoid detracting corridors and rooms, and that the expensive implementation allows selectively ignoring perception where it is inaccurate.

© 2019 Published by Elsevier Ltd.

1. Introduction

Imagine a scenario where a robotic agent is given an extended lifetime to explore an area unreachable to humans due to harsh environmental conditions. The robot is able to communicate only infrequently and messages are often subject to delay. The human employers are able to communicate which findings are interesting and which are not, even though they do not know what they are looking for. These agents will require flexibility to deal with a larger number of obstacles than a human programmer could prepare them for and therefore require a general intelligence, or ability to create solutions to their own problems.

Current approaches to solve such problems include reinforcement learning (RL) (Gosavi, 2009; Sutton & Barto, 2018) and deep reinforcement learning (DRL) (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017; Lecun, Bengio, & Hinton, 2015; Li, 2017; Schmidhuber, 2015). Reinforcement learning takes place in a setting where an agent interacts with the environment to maximise the sum of rewards where its behaviour is based on a policy which maps the agent's observation, e.g. a pixel-map, to an external action, e.g. one step north. Since the space of possible observations may be large or ambiguous, additional mechanisms for perceptual processing are typically used. Active perception, defined as actions to increase the information content of sensory data (Gibson, 1966), or more broadly as modelling and control strategies for perception (Bajcsy, 1988), has been used in several reinforcement learning studies. For example, special actions can be used for recording additional information to solve partially observable environments (Crook, 2006; Whitehead & Ballard, 1990) or for repositioning sensors to better recognise patterns in the environment (Mnih, Hess, Graves, & Kavukcuoglu, 2014; Shibata, Nishino, & Okabe, 2001). DRL methods use deep neural networks to provide additional mechanisms for perceptual processing. Among these approaches, LSTM networks that learn the value of an action in a particular memory state (Bakker, 2002; Hausknecht & Stone, 2015; Sorokin, Seleznev, Pavlov, Fedorov, & Ignateva, 2015; Wierstra, Forster, Peters, & Schmidhuber, 2010) have been applied to partially observable environments and compared favourably over other methods, including Elman networks (Lin & Mitchell, 1993) and the non-recurrent Deep Q-Networks (DQN) (Mnih et al., 2015) supplied with past sensor signals as inputs.

Despite their successes, a limitation of current RL agents is that they do not allow self-modification and self-evaluation; they lack the ability to tune their own learning mechanisms to the requirements of the environment, meta-learning or learning to

[*] Corresponding author.
E-mail addresses: D.M.Bossens@soton.ac.uk (D.M. Bossens), N.C.Townsend@soton.ac.uk (N.C. Townsend), ajs502@soton.ac.uk (A.J. Sobey).

https://doi.org/10.1016/j.neunet.2019.03.006
0893-6080/© 2019 Published by Elsevier Ltd.

learn. The generality of such approaches is reduced due to the inherent inductive bias in a single learning algorithm (Mitchell, 1980; Wolpert & Macready, 1997). To address the limits of specialised AI approaches, research into Artificial General Intelligence (AGI) or Human-level artificial intelligence (Adams et al., 2012; Baum, Goertzel, & Goertzel, 2011; McCarthy, 2007; Nilsson, 2005) is making efforts towards learners with a generality greater than or equal to humans. To achieve AGI, various different approaches have been proposed. Symbolic approaches (Anderson et al., 2004; Choi & Langley, 2018; Kieras & Meyer, 1997; Laird, 2012; Lenat, Guha, Pittman, Pratt, & Shepherd, 1990; Shapiro & Rapaport, 1992) aim to emulate the cognition by allowing ability to reason about symbols, high-level discrete concepts. Subsymbolic or emergentist approaches assume that high-level concepts will emerge from elementary processes. Some such approaches attempt to reverse engineer the brain (Hawkins, 2004; Karnowski, Arel, & Rose, 2010), while others use a developmental robotics (Asada et al., 2009; Sandini, Metta, & Vernon, 2007; Zeng, Tham, Badgero, & Weng, 2002) approach. The latter are heavily inspired by reinforcement learning with intrinsic motivation (Schmidhuber, 1991; Singh, Barto, & Chentanez, 2004; Storck, Hochreiter, & Schmidhuber, 1995), lifelong reinforcement learning (Bengio, Louradour, Collobert, & Weston, 2009; Ring, 1994; Silver, Yang, & Li, 2013; Thrun & Mitchell, 1995), and developmental psychology, e.g. Piaget (1952). Still other sub-symbolic approaches, called end-to-end RL, have emphasised that a variety of human-like functions can emerge even from utilising a single neural network for deep reinforcement learning (Shibata, 2017). Hybrid systems (Cassimatis, 2002; Cassimatis, Trafton, Bugajska, & Schultz, 2004; Franklin, Madl, Mello, & Snaider, 2014; Goertzel et al., 2013, 2011; Sun & Zhang, 2004) combine strengths of both approaches by including reasoning on both the symbolic and subsymbolic level but their flexibility is limited due to the difficulty of communicating between levels. Unlike previously mentioned approaches, the universalist approach (Hutter, 2007; Nivel et al., 2013, 2014; Schmidhuber, 2004, 2007; Wang, 2007) explicitly addresses the inductive bias: universalist learners employ a meta-algorithm which constructs solution methods to the various challenges in the environment.

One class of universalist learner which allows a general mechanism for shifting inductive bias as well as the ability to learn from quantitative rewards is self-modifying policies (SMPs), special kinds of reinforcement learners which learn how to modify their own policies (Everitt, Filan, Daswani, & Hutter, 2016; Orseau & Ring, 2011; Schmidhuber, 2007; Schmidhuber, Zhao and Schraudolph, 1997). This is achieved by including instructions in the action set which modify the current policy, rather than just external actions. SMPs therefore have a meta-learning capability with an unlimited number of meta-levels, since the actions that modify the policy also modify the use of actions that modify the policy. They can mimic, expand or integrate other algorithms, taking any algorithmic routine as a single instruction and learning when to use it. Additionally, SMPs are suitable for non-episodic learning; this contrasts to traditional reinforcement learners which assume knowledge of certain terminal states after which a similar task will occur, and, in making this assumption, their memory can be safely reset as if no previous history has occurred. SMPs have proved to be useful for learning in complex domains such as multi-agent systems (Schmidhuber & Zhao, 1997; Zhao & Schmidhuber, 1998), partially observable environments (Schmidhuber, 1999; Schmidhuber & Zhao, 1997; Zhao & Schmidhuber, 1998), noisy environments (Zhao, 2002), as well as continual learning, solving problems of increasing complexity presented sequentially across the lifetime (Schmidhuber, Zhao and Wiering, 1997).

However, current practical SMPs have limited perception capabilities, they do not integrate sub-symbolic routines which would allow for pattern recognition and function approximation, or learning operations which improve discrimination between stimuli. In Schmidhuber and Zhao (1997), Schmidhuber, Zhao and Schraudolph (1997) and Zhao and Schmidhuber (1998), special active perception instructions are implemented for Incremental Self-improvement (IS) which check for particular, user specified, objects. This approach is limited because it assumes the objects that may appear are known in advance, reducing their generality. In addition the exact implementation of these instructions is not provided and no modifiable perception module is implemented which would adaptively categorise objects. Other implementations of IS (Schmidhuber, 1995, 1999; Schmidhuber, Zhao, & Wiering, 1996) rely on a working memory which may be used to store and manipulate information about previous observations numerically. Although this is more general it can be cumbersome to learn useful perceptual routines this way. For example, to achieve a procedure similar to a single forward pass of a neural network, a relatively long sequence of instructions must be learned, and this does not include the utilisation of the resulting output. This difficulty may be the source of the observations reported in Schmidhuber (1999) that the learning curve of IS is step-wise. More efficient perceptual routines could potentially make the learning curve continuous and increase performance.

The above discussion on DRL has highlighted that, due to the particular algorithmic assumptions on the perceptual system, learners may fail, either completely or for particular spatio-temporal patterns, when they are subjected to atypical environments where such assumptions are not met. Agents that would be able to flexibly use and modify their perceptual system may therefore have an improved performance in atypical environments, especially when given an extended lifetime to learn how to learn. Therefore, this paper explores *active adaptive perception*. Active adaptive perception is :

- An active form of adaptive perception: adaptive perception is the long-term adaptation of perceptual systems to environmental demands. To be an active adaptive perceiver then means to be able to decide how and when to modify the perceptual system, based on environmental demands.
- Active perception: based on Bajcsy, Aloimonos, and Tsotsos (2017), *an agent is an active perceiver if it knows why it wishes to sense, and then chooses what to perceive, and determines how, when and where to achieve that perception.*

To address active adaptive perception, a generic SMP learning architecture is proposed which uses special instructions to improve its perceptual architecture and to adapt the learning and use of its perceptual operations based solely on the reinforcement signals it receives. The proposed architecture is intended for complex, long-term reinforcement learning problems. To demonstrate the approach (a) a computationally cheap implementation is presented which improves its perception by modifying the weights, topology and use of a neural network perception module; (b) a more expensive implementation which allows the learner to choose situations and parameters for training and utilising a deep recurrent Q-network. Illustrative experiments are performed on non-episodic partially observable mazes comparing the implementation to DRQN (Hausknecht & Stone, 2015), an off-policy deep reinforcement learner using experience replay, and Incremental Self-improvement (Schmidhuber, 1999), representing traditional SMPs. Approaches similar to this implementation are expected to improve training and construction of neural networks.

2. Learning with limited knowledge and sparse feedback

A general learner must be able to consistently rank highly across all problems. An example where general learners demonstrate some benefits are atypical environments, where traditional RL assumptions are not valid: terminal states do not exist or are unknown, the environment is partially observable, and the rewards are incurred on a sparse basis with the possibility of not getting feedback at all. Further, unlike some types of Partially Observable Markov Decision Process solvers, such as those in Kaelbling, Littman, and Cassandra (1998), Shani, Pineau, and Kaplow (2013) and Silver and Veness (2010), but similar to many other RL methods for partially observable environments, the underlying state space is not known.[1] The environments considered here are atypical problems, where it is expected that general solvers will outperform specialist equivalents.

The maze setting considered in Schmidhuber (1999), and investigated in this paper, is an example of an atypical environment. The learner has a lifetime going from $t = 0$ to $t = T$ without any interruptions. The learner initially wanders around without knowing what its goal is. At each time-step, it obtains an observation about whether the four adjacent locations in a Von Neumann neighbourhood are free or blocked and selects an operator from the set of external actions $\mathcal{A}^E = \{\texttt{north}, \texttt{east}, \texttt{south}, \texttt{west}\}$. After taking such an external action, a reward of 1.0 is given when the learner finds the goal position, otherwise a reward of 0.0 is given. The learners task is to maximise the cumulative reward. Whenever a goal is obtained, the learner is reset to a starting position. However, the environment appears as *non-episodic* to the learner since it has no knowledge of this inherent episodic structure: the goal location is not noted as a terminal state, the memory is not reset after reaching the goal, and there is no artificial time-out to reset the learner when the learner does not reach the goal. Instead, the environment appears to the agent as a single history from $t = 0$ to an unknown $t = T$. Another source of ambiguity is that the learner is reset immediately to a start position without recording an observation at the goal location.

If the above maze setting is extended to larger mazes with similar obstacle density, then this problem is challenging: detracting corridors and rooms, an initially faulty policy, a sparse reward structure, and a non-episodic setting without time-outs, the learning agent may get stuck in a bad region of the maze and experience thousands of steps without any rewards. This is significantly different from other partially observable environments investigated in deep reinforcement learning papers, such as
T-maze experiments (Bakker, 2002), the 89-state maze (Wierstra et al., 2010), Atari experiments (Hausknecht & Stone, 2015) and the Invisible Target Capture Task (Shibata & Goto, 2013). Those experiments are comparatively easy in the sense that: (a) the agent has knowledge of terminal states and the memory is reset at the start of the episode; (b) to avoid getting stuck without feedback, there is an artificial time-out such that, after a certain number of time steps without reaching the goal, the agent is reset to the initial state; (c) there is no corridor from which it is difficult to escape, instead the space is open or there is a single path; (d) reward structure is more dense, for example, by giving feedback about whether or not the step lead closer to goal. Nevertheless, those experiments have difficulties not addressed in the present maze setting: for example, Atari experiments and the Invisible Target Capture Task have complex dynamics and a large state-space.

[1] These methods are a subset of Partially Observable Markov Decision Process solvers which estimate the environment state, which is unknown, from the known observations, usually using the "belief state" $p(s_t|h_t)$, the probability of the environment's state given the history of observations and actions. In the below maze, each (x, y)-coordinate's probability could be estimated from the previous Von Neumann neighbourhood observations and the previous actions. Such an estimation procedure implies that the designer has specific domain knowledge of the true environment; in the maze example, the domain knowledge is that x and y are the main variables of the state space, even though their exact values are never known.

3. Generic architecture for active adaptive perception

This section proposes a generic architecture and an exemplar implementation, illustrated in Fig. 1. The generic architecture consists of four basic components: an instruction module, an evaluation module, a working memory and a perception module. It serves as an abstract template for learners with active adaptive perception; the way the architecture is implemented may alter efficiency but not the property of active adaptive perception. In the exemplar implementation, the perception module is implemented as a neural network with an evolvable representation as in NEAT (Stanley & Miikkulainen, 2002), whereas other modules are implemented according to Incremental Self-improvement (Schmidhuber, 1999). After providing an algorithmic overview and the rationale, the remainder of the section describes each module and its corresponding implementation in detail.

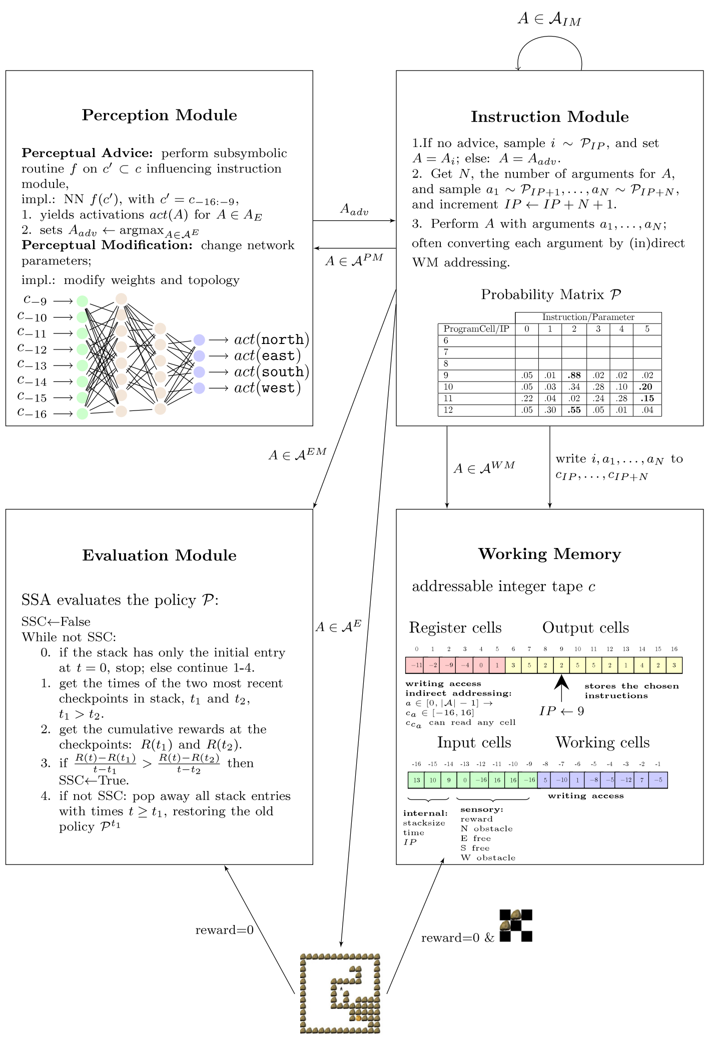
3.1. Rationale

To achieve active adaptive perception, an architecture of at least two components is proposed: first, a universalist meta-algorithm, called the instruction module, utilises elementary instructions to construct perceptual modification algorithms and to selectively apply the perceptual apparatus; second, a sub-symbolic system, called the perception module, is being selectively called by some of the instruction module's instructions, either to make long-term modifications to the sub-symbolic system, perceptual modification, or to utilise the patterns detected by the perception module to temporarily influence which instructions are generated by the instruction module, perceptual advice. Using these instructions, the system can then successfully learn when and how to request further perceptual processing and when and how to make long-term modifications to the perceptual system, thereby achieving active adaptive perception.

Due to fitting the above description, a natural candidate for the instruction module is the self-modifying policy (SMP) learner, a special reinforcement learning policy. A reinforcement learning policy $\mathcal{P} : \mathcal{O} \rightarrow \mathcal{A}$ outputs actions $A \in \mathcal{A}$ based on the agent's observation $o \in \mathcal{O}$, thereby maximising a particular utility-function. An SMP is a special kind of reinforcement learner which includes in $\mathcal{A}$ instructions that modify $\mathcal{P}$, and possibly various other instructions for performing computations. If such a policy successfully learns when and how to use self-modification instructions, such a policy learns how to generate changes to itself. This allows learning even when typical reinforcement learning assumptions are not met, and therefore provides improved generality. Moreover, if such a policy successfully learns when and how to use perceptual modification and perceptual advice instructions, according to their above definition, this will achieve active adaptive perception.

3.2. Algorithmic overview

A learner, $\mathcal{L}$, is put in an environment, $\mathcal{E}$, with the aim of maximising the sum of rewards. $\mathcal{L}$ seeks out rewards by outputting an instruction a from a user-defined set of instructions $\mathcal{A}$.



Perception Module

Perceptual Advice: perform subsymbolic routine f on $c' \subset c$ influencing instruction module,
impl.: NN $f(c')$, with $c' = c_{-16:-9}$,
1. yields activations $act(A)$ for $A \in A_E$
2. sets $A_{adv} \leftarrow \text{argmax}_{A \in \mathcal{A}^E}$

Perceptual Modification: change network parameters;
impl.: modify weights and topology

Instruction Module

1.If no advice, sample $i \sim \mathcal{P}_{IP}$, and set $A = A_i$; else: $A = A_{adv}$.
2. Get N, the number of arguments for A, and sample $a_1 \sim \mathcal{P}_{IP+1}, \dots, a_N \sim \mathcal{P}_{IP+N}$, and increment $IP \leftarrow IP + N + 1$.
3. Perform A with arguments $a_1, \dots, a_N$; often converting each argument by (in)direct WM addressing.

Probability Matrix $\mathcal{P}$

	Instruction/Parameter					
ProgramCell/IP	0	1	2	3	4	5
6						
7						
8						
9	.05	.01	**.88**	.02	.02	.02
10	.05	.03	.34	.28	.10	**.20**
11	.22	.04	.02	.24	.28	**.15**
12	.05	.30	**.55**	.05	.01	.04

Evaluation Module

SSA evaluates the policy $\mathcal{P}$:

```
SSC←False
While not SSC:
    0. if the stack has only the initial entry at $t = 0$, stop; else continue 1-4.
    1. get the times of the two most recent checkpoints in stack, $t_1$ and $t_2$, $t_1 > t_2$.
    2. get the cumulative rewards at the checkpoints: $R(t_1)$ and $R(t_2)$.
    3. if $\frac{R(t)-R(t_1)}{t-t_1} > \frac{R(t)-R(t_2)}{t-t_2}$ then SSC←True.
    4. if not SSC: pop away all stack entries with times $t \geq t_1$, restoring the old policy $\mathcal{P}^{t_1}$
```

Working Memory

addressable integer tape c

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
−11	−2	−9	−4	0	1	3	5	2	2	5	5	2	1	4	2	3

writing access
indirect addressing:
$a \in [0, |\mathcal{A}| - 1] \rightarrow$
$c_a \in [-16, 16]$
c_{c_a} can read any cell

-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
13	10	9	0	−16	16	16	−16	5	−10	1	−8	−5	−12	7	−5

Fig. 1. Diagram of the generic architecture for active adaptive perception, and its current implementation on the maze problem. Based on the current instruction pointer as an internal state, the instruction module generates an instruction and its arguments. Working memory elements, integers in [−16, 16], are then used to process the arguments for context-sensitive instruction execution. The instruction is then performed, calling the evaluation module to perform SSA, the perception module to modify it or to request an advised action, the working memory to make historical notes, the instruction module to self-modify, or an external action in the environment. When perceptual advice is requested, the perception module's neural network outputs an advised action after taking the input cells' contents, normalised in [−1, 1], as inputs. Input cells are the current reward, the binary observation bits indicating whether north, east, south, and west are free positions or obstacles, and internal variables for disambiguating the state based on the history, namely the time, the stacklength and the instruction pointer. Note that (a) a simplified representation is given because the number of working memory cells and program cells is larger in the experiments; (b) in the SMP-DRQN implementation, the inputs to the perception module is the history of observations instead of all current input cells, and additional perceptual modifications are done on a set of useful experiences.

Instructions are similar to functions used in programming languages which take several arguments as inputs and then perform some computations based on these arguments. The following subsection describes how the short-term behaviour, which is structured into instruction cycles, of $\mathcal{L}$ leads to the long-term process of active adaptive perception, due to the usage of various instruction types.

A single instruction cycle. A single instruction cycle works as follows. The agent, with its N sensors, receives an observation $o \in \mathcal{O} \subset \mathbb{R}^N$ from $\mathcal{E}$ and writes it directly to the working memory elements reserved for observation. Based on a fixed subset of working memory cells, not necessarily the same as observation elements, the instruction module, based on the instruction module parameters $\mathcal{P}$, generates an instruction $A \in \mathcal{A}$ and its arguments $a_1, \dots, a_N$, if that instruction has any. The execution of A results in interactions between modules (internal actions) or between $\mathcal{L}$ and $\mathcal{E}$ (external actions $A \in \mathcal{A}^E$); different types of interactions are described in the following paragraph. As in reinforcement learning, a critic in the environment $\mathcal{E}$ sends a reward $r \in \mathbb{R}$, usually only when an external action is taken. The cycle ends after $\mathcal{L}$ has processed the reward in the evaluation module.

Instruction types. Interactions between modules or between $\mathcal{L}$ and $\mathcal{E}$ are encoded in the instructions; the following types of instructions are the minimal requirement for active adaptive perception:

- External actions ($A \in \mathcal{A}^E$): interact with $\mathcal{E}$ to obtain rewards
- Self-modification ($A \in \mathcal{A}^{IM}$): modify the instruction module parameters $\mathcal{P}$.
- Working memory manipulation ($A \in \mathcal{A}^{WM}$): change the working memory based on the sensory inputs and the current working memory
- Perceptual advice ($A \in \mathcal{A}^{PM}$): based on the current working memory a part of the perception module computes outputs which influence instruction generation.
- Perceptual modification ($A \in \mathcal{A}^{PM}$): modify the perception module's parameters, changing the way perceptions form and interact with the instruction module.
- Evaluation ($A \in \mathcal{A}^{EM}$): call the evaluation module to evaluate changes to the instruction module and the perception module

Steps involved in the learning process. All instructions must be regularly used which results in the following learning steps:

1. The learner starts with a minimally biased instruction and perception module;
2. Intermittently, changes to instruction generation and the perceptual processes are made;
3. Once the evaluation module is called, the changes are accepted or rejected based on evidence of their contribution to reward intake;
4. The instruction module thus learns when to execute the instructions but also how; the meaning of instructions is optimised as the arguments supplied to the instruction is changed.
5. This leads to optimisation of the interaction between the various modules. One of the consequences is perceptual learning, which may be divided into two processes:
    - long-term parameter changes: similar to traditional sub-symbolic learners, the best parameters for a given perceptual operation are learned.
    - active adaptive perception: learning how to modify and use the perception module.

3.3. Instruction module

The instruction module organises the interactions with the environment but also with the different modules of the architecture by utilising a user-defined instruction set $\mathcal{A}$, a set of operations which includes external actions which involve interacting with the environment, e.g. moving one step north, grabbing an object, or applying sensory mechanisms; and internal operations to enable memory, learning and inference. The mechanism of the instruction module is to continuously generate instructions based on the current instruction module parameters $\mathcal{P}$ and a set of working memory variables.

Implementation: probability matrix $\mathcal{P}$. The learner's policy $\mathcal{P}$ consists of a number of m program cells $\mathcal{P}_i$ ($i = ProgramStart, \dots, ProgramStart + m - 1$) each of which represents a discrete modifiable probability distribution over the integers $\{0, \dots, |\mathcal{A}| - 1\}$ initialised to a uniform distribution but subject to change due to self-modification instructions. Using the instruction pointer IP, a special working memory variable that points to the current program cell, an instruction cycle consists of sampling an integer $j \sim \mathcal{P}_{IP}$ representing an instruction $A_j \in \mathcal{A} = \{A_0, A_1, \dots, A_{|\mathcal{A}|-1}\}$. After checking how many arguments, N, are required for executing A_j, integer arguments $a_1, \dots, a_N$ are generated according to the distributions of the following program cells $\mathcal{P}_{IP+1}, \dots, \mathcal{P}_{IP+N}$. After executing the instruction and its arguments, a new cycle starts with $IP \leftarrow IP + N + 1$. Instructions include various external actions but also internal operations. For self-improvement, the learner uses self-modification instructions `incP` and `decP` which increase and decrease the probability of a chosen program cell $\mathcal{P}_i$ by a chosen amount, respectively. Evaluation is initiated by `endSelfMod` which ends the current modification sequence and starts the evaluation of the latest changes made to the instruction module and the perception module. External actions are application-dependent, such as `north`, `east`, `south` and `west` in maze-problems. The working memory is manipulated using reading, writing and arithmetic operations and instructions that change the IP similarly determine the state of the learner. Perceptual modification and perceptual advice instructions are explained in Section 3.6, while a complete instruction set is given in Table 1.

3.4. Evaluation module

The function of the evaluation module is to determine if changes to the instruction module and the perception module are beneficial by considering evidence of how self-modifications relate to reward intake.

Implementation: Success story algorithm. To allow a learner to incrementally improve its performance with minimal assumptions on the environment, an empirical evaluation method called the Success Story Algorithm (SSA) is used which maintains only those incremental modifications that lead to long-term reward acceleration. At time points called checkpoints initiated by the instruction `endSelfMod`, the learner performs an evaluation of the current self-modification sequence (SMS). The learner can adapt to tasks with atypical reward structures because it can determine the frequency of `endSelfMod`, learning how much time is required to reliably evaluate a series of modifications. The evaluation is done using the Success Story Criterion (SSC),

$$\frac{R(t) - R(t_2)}{t - t_2} > \frac{R(t) - R(t_1)}{t - t_1}, \tag{1}$$

where $R(t) = \sum_{\tau=0}^{t} r(\tau)$ is the cumulative reward, t is the current time and t_2 and t_1 are the most and second-most recent checkpoints, respectively. Thus the SSC asserts whether the reward

Table 1
List of instructions used for the instruction set $\mathcal{A}$ in the SMP learners. Instructions are divided in categories based on the module it directly affects: *E* for environment, *PM* for perception module, *IM* for instruction module, and *WM* for working memory. The SMPs included in the experiments used a different subset of $\mathcal{A}^{PM}$, the instructions relevant for active adaptive perception, and the set $\mathcal{A} \setminus \mathcal{A}^{PM}$ are instructions commonly used in Incremental Self-improvement. **Function and operator definitions:** c is the working memory tape, often indexed by double/indirect-addressing; $layer(i)$ obtains the layer index of node i; $narr(a, [b, c])$ performs a narrowing conversion from $a \in [0, |\mathcal{A}|-1]$ to an integer in $[b, c]$; $switch(from, to)$ switches *from* and *to* when $from > to$ or aborts the instruction when $from = to$; $\mathcal{N}(\mu, \sigma)$ is the normal/Gaussian distribution; $clip(a; [b, c])$ clips a to an integer in the range $[b, c]$. $a//b$ returns $sign(a)*MaxInt$ if $b = 0$ and integer division otherwise; a mod b returns a if $b = 0$ and $a - b * floor(a/b)$ otherwise.

Instruction	Type	Explanation
`north`	$\mathcal{A}^E$	Take one step north
`east`	$\mathcal{A}^E$	Take one step east
`south`	$\mathcal{A}^E$	Take one step south
`west`	$\mathcal{A}^E$	Take one step west
`getOutput()`	$\mathcal{A}^{PM}$	Forward inputs $c_{-16:-9}$ through the perception module network, yielding activations $act(A)$ for all $A \in \mathcal{A}^E$. Set $A_{adv} \leftarrow \text{argmax}_{A \in \mathcal{A}^E} act(c')$; Next cycle the instruction module will execute A_{adv}.
`doQuntil`(a_1, a_2, a_3)	$\mathcal{A}^{PM}$	If $looping = True$ or $t < replayStart$ return; else, set $looping \leftarrow True$, the termination experience $term \leftarrow \text{E}_{a_1}$ as the a_1'th element of the experience set E, the maximal number of looping cycles $until \leftarrow narr(a_2, [1, unroll/2])$, and $\epsilon \leftarrow a_3 * .005$. The next cycles, the DRQN network outputs as the activations $act(A)$ the Q-values $Q(s, a)$ for all $A \in \mathcal{A}^E$ with s denoting the history of observations, and then the ϵ-greedy strategy, with the self-chosen ϵ, selects the next external action. The loop is terminated when the current experience is *term* or when *until* time steps have passed.
`weightChange`(a_1, a_2)	$\mathcal{A}^{PM}$	Add a copy of the current network to the stack $\mathcal{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes} - 1])$, $j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes} - 1])$; set $w_{ij} \leftarrow clip(w_{ij} + \mathcal{N}(0, \sigma_w); range_w)$ with $range_w = [-50, 50]$ and $\sigma_w = 5.50$.
`addNode`(a_1, a_2)	$\mathcal{A}^{PM}$	Add a copy of the current network to the stack $\mathcal{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes} - 1])$, $j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes} - 1])$; perform $switch(i, j)$; if $layer(j) > layer(i) + 1$ then delete the old connection $(from = i, to = j, w = w_{ij})$, add a new node k in layer $layer(i) + 1$ and add connections $(from = i, to = k, w = 1)$ and $(from = k, to = j, w = w_{ij})$.
`addConnection`(a_1, a_2)	$\mathcal{A}^{PM}$	Add a copy of the current network to the stack $\mathcal{S}$. set $i \leftarrow narr(c_{c_{a_1}}, [0, n_{nodes} - 1])$, $j \leftarrow narr(c_{c_{a_2}}, [0, n_{nodes} - 1])$; perform $switch(i, j)$; create a new connection gene $(from, to, w)$ with $w \sim \mathcal{N}(0, \sigma_w)$
`setExperience`(a_1)	$\mathcal{A}^{PM}$	If $t < replayStart$, return; else, add the current value of E_{a_1} to the stack $\mathcal{S}$, the a_1'th element of the experience set, to the stack and replace it with the current experience: $\text{E}_{a_1} \leftarrow (o, A, r, o')$, with o the previous observation, A the previous external action, r the current reward, and o' the current observation.
`incP`(a_1, a_2, a_3)	$\mathcal{A}^{IM}$	Push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathcal{S}$. Then, set $\mathcal{P}_{c_{a_1}, c_{a_2}} \leftarrow 1 - .01 * c_{c_{a_3}} * (1 - \mathcal{P}_{c_{a_1}, c_{a_2}})$, with $c_{a_1} \in \{0, \ldots, \vert\mathcal{A}\vert-1\}$; $\mathcal{P}_{c_{a_1}, i} \leftarrow .01 * c_{c_{a_3}} * \mathcal{P}$ for all $i \in \{0, \ldots, \vert\mathcal{A}\vert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1}, i} < minP = 0.0005$ for any $i \in \{0, \ldots, \vert\mathcal{A}\vert-1\}$
`decP`(a_1, a_2, a_3)	$\mathcal{A}^{IM}$	Push the current probability distribution $\mathcal{P}_{c_{a_1}}$ to the stack $\mathcal{S}$. Then, set $\mathcal{P}_{c_{a_1}, c_{a_2}} \leftarrow .01 * c_{c_{a_3}} * \mathcal{P}_{c_{a_1}, c_{a_2}}$, with $c_{a_1} \in \{0, \ldots, \vert\mathcal{A}\vert-1\}$; $\mathcal{P}_{c_{a_1}, i} \leftarrow \mathcal{P}_{c_{a_1}, i} * (1 - .01 * c_{c_{a_3}} * \mathcal{P}_{c_{a_1}, c_{a_2}})/(1 - \mathcal{P}_{c_{a_1}, c_{a_2}})$ for all $i \in \{0, \ldots, \vert\mathcal{A}\vert-1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1}, i} < minP = 0.0005$ for any $i \in \{0, \ldots, \vert\mathcal{A}\vert-1\}$.
`endSelfMod()`	$\mathcal{A}^{EM}$	Evaluate the current self-modification sequence with SSA
`jumpHome()`	$\mathcal{A}^{WM}$	Set $IP \leftarrow ProgramStart$
`jumpEq`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	If $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP \leftarrow c_{c_{a_3}}$.
`jumpLower`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	If $c_{c_{a_1}} = c_{c_{a_2}}$, set $IP < c_{c_{a_3}}$.
`add`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} + c_{c_{a_2}}; [MinInt, MaxInt])$
`sub`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} - c_{c_{a_2}}; [MinInt, MaxInt])$
`mult`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} * c_{c_{a_2}}; [MaxInt, MaxInt])$
`div`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} // c_{c_{a_2}}; [MinInt, MaxInt])$
`rem`(a_1, a_2, a_3)	$\mathcal{A}^{WM}$	$c_{c_{a_3}} \leftarrow clip(c_{c_{a_1}} \bmod c_{c_{a_2}}; [MinInt, MaxInt])$
`mov`(a_1, a_2)	$\mathcal{A}^{WM}$	$c_{c_{a_2}} \leftarrow c_{c_{a_1}}$
`init`(a_1)	$\mathcal{A}^{WM}$	$c_{a_2} \leftarrow a_1 - ProgramStart - 2$
`inc`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} + 1; [MinInt, MaxInt])$
`dec`(a_1)	$\mathcal{A}^{WM}$	$c_{c_{a_1}} \leftarrow clip(c_{c_{a_1}} - 1; [MinInt, MaxInt])$

Note: some operations yield invalid addresses or numbers according to rules of syntactical correctness (cf. Schmidhuber, 1999); if these conditions are not met the operation does nothing except for the usual increments to the instruction pointer *IP*.

intake has accelerated since $t_2 > t_1$. If this is true, then modifications made in $[t_1, t_2]$ will be maintained, otherwise the current modifications are removed and the old instruction module and perception module before t_1 are restored, recursively, until the SSC is met. The complete list of modifications that survived the SSC is maintained in the stack $\mathcal{S}$. The recursive procedure of popping entries that do not yield reward acceleration is illustrated in Algorithm 1.

3.5. Working memory

A working memory is used to store and manipulate information from various parts of the learning structure as well as the environment. Variables in the working memory are updated regularly or at self-chosen times by the instruction module and the environment. The working memory provides other modules with historic information, contributing to non-Markovian learning and decision-making.

Algorithm 1 Success story algorithm for evaluating self-modification sequences (SMSs). Note that when $sp = 1$, the top entry is compared to the initial entry $\mathcal{S}[0] = (t = 0, R = 0, \mathcal{X} = \emptyset, first = 0, address = \emptyset)$.

```
$sp \leftarrow length(\mathcal{S}) - 1$
while True do
    if $sp = 0$ then
        break;                          ▷ no modifications left; SSC satisfied
    else
        $i \leftarrow \mathcal{S}[sp].first$;                    ▷ first index of the top SMS
        $j \leftarrow \mathcal{S}[i-1].first$;      ▷ first index of the second-top SMS
        if $\frac{R(t)-\mathcal{S}[i].R}{t-\mathcal{S}[i].t} > \frac{R(t)-\mathcal{S}[j].t].R}{t-\mathcal{S}[j].t}$ then
            break;                      ▷ reward accelerates; SSC satisfied
        else
            while $sp \geq i$ do
                if $\mathcal{S}[sp].\mathcal{X}$ is a modification of $\mathcal{P}$ then
                    $\mathcal{P}_{address_1} = \mathcal{S}[sp].\mathcal{X}$;    ▷ restore probability vector at index $address_1$.
                else
                    restore the parameters of the perception module using $\mathcal{S}[sp].\mathcal{X}$.
                delete $\mathcal{S}[sp]$;
                $sp \leftarrow sp - 1$;
```

Implementation: addressed integers. The implementation of the working memory is a number of storage cells, each with a unique address in $[Min, Max]$ store integer values in $[-MaxInt, MaxInt]$, initialised randomly and then changed by fixed routines or self-chosen instructions. The functionality of the working memory can be categorised into four types, illustrated in Fig. 1. Each of the cells have particular integer address to access them, and Appendix B indicates how to obtain the exact addresses used in the experiments. Input cells are special working memory cells that are updated every cycle and which store the current observation as well as other information, i.e., the instruction pointer IP, the time t, the reward r and the length of the stack $\mathcal{S}$. Output cells store the history of recent actions: when $\mathcal{P}_{IP}$ has generated an integer, this number is written to the output cell that is addressed by IP. Working cells provide more long-term memories, only being modified when special instructions in $\mathcal{A}^{WM}$ manipulate them. Register cells have the same function but additionally they are used for a process called double-indexed addressing: because $\mathcal{P}$ generates arguments in the limited range $[0, |\mathcal{A}|-1]$, the values of the register cells in $[0, |\mathcal{A}|-1]$ are used to address the entire range of working memory cells. Working memory contents are used extensively in the execution of the various instructions. For example, the instruction `add(`a_1, a_2, a_3`)` first reads the contents of the register cell at address a_1 and then fetches the value at the address c_{a_1}, as notated by $c_{c_{a_1}}$; then similarly reads $c_{c_{a_2}}$; finally, adds both $c_{c_{a_1}} + c_{c_{a_2}}$ and stores this sum on the address c_{a_3}. Similar to the above example, many other instructions, including instructions used for self-modification and perceptual modification, also use working memory contents to determine how the instructions are executed.

To illustrate how the processing of the working memory can be used for processing observations to influence external actions, an illustrative example is mentioned for the maze example of Section 2:

1. first, the agent records an observation and a reward in its input cells, indicating whether the neighbouring positions are obstacles or free spaces and whether or not it reached a goal;
2. then, it uses working memory operators to manipulate the memory, based on various cells including the working cells;
3. an execution of `jumpEq` or `jumpLower` then sets the instruction pointer IP based on working memory contents;
4. eventually, an external action is executed based on IP".

3.6. Perception module

The perception module is a modifiable sub-symbolic module whose function is to advise the instruction module, using special instructions relevant for active adaptive perception. It supplies bottom-up perception to the architecture by analysing sensory inputs and working memory variables in terms of high-level concepts to help decision-making. For example, successive layers of a neural network may process elementary visual stimuli such as edges into shapes and objects, and a configuration of free and blocked spaces in a maze may be processed in terms of a narrow corridor or a wide area. By interacting with the working memory and the instruction generation, it can influence the decisions made by the instruction module, using perceptual advice instructions. The architecture and the parameters of the perception module are subject to long-lasting modifications when the instruction module calls special perceptual modification instructions. By learning when to use which perceptual advice and perceptual modification instruction, various strategies for utilising and training the perception module will emerge from experience, active adaptive perception. Two implementations were made to demonstrate that the architecture of the perception module and the instructions for the perceptual advice and modifications can vary while still providing active adaptive perception. The first implementation would likely be more suitable for real-time environments where learning should not consume too much time, whereas the second implementation is more computationally expensive but makes better use of its experiences. The relevant instructions are displayed in Table 1, in the group $\mathcal{A}^{PM}$.

Implementation 1: NEAT neural network. The first implementation considers simple instructions to use and modify a NEAT feed-forward network. To achieve **perceptual advice** in this implementation, a special instruction `getOutput()` performs forward pass of the perception module's feedforward neural network which takes as input the eight input cells of the working memory and then outputs activations $act(a)$ for each external action. Based on these output activations an advisory action A_{adv} is selected to be executed at the next instruction cycle. A unit u, an elementary node in the network, activates its incoming activation node-input(u, t) at time t according to:

$$u(t) = type_u(\text{node-input}(u, t)), \tag{2}$$

where $type_u$ is the transfer function of u. If unit u is situated at layer ℓ, the node input is defined by:

$$\text{node-input}(u, t) = \sum_{v \in U^{l<\ell}} w_{uv} v(t) \tag{3}$$

where $U^{l<\ell}$ is the set of nodes at a layer lower than ℓ. To achieve **perceptual modification**, the learner uses computationally cheap instructions `weightChange`, `addNode` and `addConnection` to modify both topology and weights of a neural network. Each change to the network is recorded on the stack $\mathcal{S}$ such that it can be evaluated later by the evaluation module. This is done with a special representation similar to NEAT, where a neural network consists of two sets of genes. *Connection genes* are tuples of $(from, to, weight)$: *from* and *to* represent the connections input and output unit respectively and *weight*

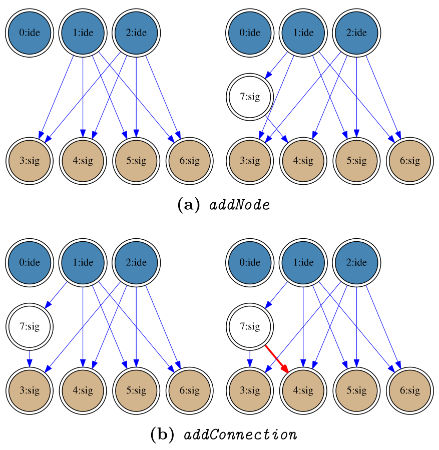


Fig. 2. Illustration of the network construction operators. For simplicity, only two input nodes are shown and the bias unit is shown without its connections. Network connections are restricted such that the layers satisfy *from* < *to*. Blue nodes indicate input units, grey–brown nodes indicate output nodes, and white nodes indicate hidden nodes. Input units use the identity function (denoted by `ide`) as a transfer function, while non-input units use the sigmoid function (denoted by `sig`). The added connection is emphasised in red bold. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

represents the interconnection weight. *Node genes* are tuples of $(type, bias, response, layer)$: *type* is the transfer function used to output at the neuron, *bias* encodes a number to be added to the activation independent of all other incoming activations , *response* is a number that the neuron multiplies with all its incoming weights and *layer* is used to adequately structure the connections. Together, the node genes and the connection genes represent the neural network which is being learned, allowing the use of constructive operators `addNode` and `addConnection`, shown in Fig. 2a and 2b, respectively. Other parameters not changed by the above instructions were fixed, with *type* being sigmoid for non-inputs and identity for inputs, and $bias = 0$ and $response = 4.92$ for all neurons. The weight-range and the response are selected based on the peas-neat implementation, cf. https://github.com/noio/peas/blob/master/peas/methods/neat.py, as they are commonly used settings.

Implementation 2: DRQN with modifiable experience set. The second implementation embeds the learning of the Deep Recurrent Q-Networks (Hausknecht & Stone, 2015), a recurrent extension to Deep Q-Networks (Mnih et al., 2015), and a modifiable experience set as its perception module. The experience set E is a database of interesting experiences which serve as goals after which network use is halted. This enables selective network usage, learning when to rely on the Q-network, as well as goal-based exploration, learning when to use which exploration rate.

While observing and acting, DQN fills a buffer $\mathcal{B}$ with experiences (o, A, r, o'), which are tuples of observation, chosen action, observed reward, and the following observation. DQN minimises a loss function

$$L(\theta) = \mathbb{E}_{(o,A,r,o')\sim U(\mathcal{B})}[(r + \gamma \max_{A'} Q(o', A'; \hat{\theta}) - Q(o, A; \theta))^2] \quad , \quad (4)$$

where experiences are sampled uniformly from the buffer $\mathcal{B}$ for the next mini-batch update, a process called experience replay; Q is the value-function for Q-learning (Watkins & Dayan, 1992); γ is the discount used to compute the discounted future cumulative reward $\sum_{i=0}^{\infty} \gamma^i r^i$; θ contains the current parameters whereas $\hat{\theta}$ contains the parameters of the target network which are updated only infrequently, increasing stability. In DRQN, the loss function is the same but the observations are passed through a recurrent network including a Long Short-Term Memory layer (Hochreiter & Schmidhuber, 1997), such that the history of observations affects the internal state of the learner. The experience replay thus is modified in DRQN to randomly sample a history of *unroll* consecutive experiences rather than a single experience.

Perceptual advice in the second implementation consists of a single instruction `doQuntil`. When the instruction module performs the `doQuntil` instruction the DRQN network takes the observation o as its only input, with no other working memory variables, and outputs the Q-values of the different actions from which it determines the advised action. An ϵ-greedy strategy is used such that with probability $1 - \epsilon$ the chosen action is $A_{adv} = \text{argmax}_{A' \in \mathcal{A}^E} Q(o, A')$ whereas with probability ϵ a random action is chosen. This is done repeatedly until a self-chosen experience tuple $e_{a_1} = (o, A, r, o')$ is achieved or until a number of self-chosen time-steps are reached without matching e_{a_1}. The arguments of the `doQuntil` instruction determine three important parameters: a_1 determines which experience is taken from the experience set E to end the loop, a_2 determines the number of maximal steps of the loop, and a_3 determines the exploration ϵ. **Perceptual modification** consists of two instructions. The first, `trainReplay`, is the typical experience replay procedure as is performed in DRQN, but with the added flexibility that the instruction module determines the batch size. This instruction is not followed by pushing the network modification onto the stack $\mathcal{S}$ since this instruction already combines updating with an immediate evaluation. The second, `setExperience`, adapts the set of experiences E by replacing the experience at index a_1, E_{a_1}, by the current experience. This instruction then pushes this modification to the stack to allow the evaluation module to perform long-term evaluation of the new E. Note that E is initialised with experiences randomly drawn from the experience buffer just before the replay-start at $t = 50000$.

3.7. How the exemplar learns

Instructions `incP` and `decP` modify the probability of a particular entry $\mathcal{P}_{ij}$, and normalisation is then performed ensuring $\sum_k \mathcal{P}_{ik} = 1$. This results in a change of the probability distribution of instructions for a given program cell *i*. In turn, this changes the system's response to the internal state $IP = i$, a variable changed by the various `jump` instructions and incremented by executing instructions. The probability changes affect not only external but also the internal behaviours due to the choice of instructions and arguments for working memory manipulation, evaluation, perceptual advice, perceptual modification, and self-modification. For self-modification instructions, this leads to a self-referential recursion, meta-meta-...-learning: changes to $\mathcal{P}$ affect how $\mathcal{P}$ will be changed, and so on. In particular, self-modifications may affect the probability of `incP`, `decP` or its arguments for a given internal state *IP* and given working memory state, implying a context-sensitive learning of (a) self-modification probability; (b) which program cell should be modified; (c) which entry in the program cell's distribution should be incremented or decremented; (d) how large the increment or decrement should be. Because the Success Story Algorithm repeatedly evaluates previous self-modifications and maintains only those self-modifications that result in lifetime reward acceleration, early self-modifications result in better generators of self-modifications later on; Assuming the instructions cover all aspects of learning and behaviour, this means $\mathcal{P}$ improves itself.

4. Experimental set-up

All experiments took place in the exemplary non-episodic maze setting explained in Section 2. However, it is assumed that each action consumes one time-step and computational processes do not consume time, diverging in this respect from Schmidhuber (1999). This facilitates interpretation and also comparison to traditional reinforcement learning which have the same assumption. The section further justifies the selection of experimental conditions and learners used in the experiments.

4.1. Experimental conditions

There are four experimental conditions based on the dimensions *easy* vs. *difficult* and *fixed* vs. *random*. Easy problems have shorter optima, 4–8 steps vs. 11–30 for difficult problems, often with lower ambiguity and fewer free spaces to get lost in. The difficulty is used to test the hypothesis that self-modifying policies are beneficial when environments have higher ceilings of performance. Higher ceilings are defined as a higher potential to increase the reward intake speed compared to a random learner which for every 10^4 time steps had 30–120 rewards for easy and 1–10 rewards for difficult problems. The second dimension, fixed vs. random, describes what happens after goal achievement, concretely whether or not the next starting position is fixed or chosen randomly. Ten easy and ten difficult mazes were generated according to Algorithm 2, found in the Appendix A, on a grid of dimensions 13×13. The resulting mazes had a variety of features: wide open spaces, narrow straight corridors and intersecting corridors which results in central decision points.

For easy problems, each learner was given a lifetime of 5 million time steps because initial experiments suggested learners had converged by then. For difficult problems, optimal path lengths increased approximately four times, and the actual path lengths, and thus the time to learn from rewards, relates exponentially to the optimal path length due to the increase in possible misleading explorations. 80 million time steps were judged to be a reasonable number without excessive computational expense. Experiments on the difficult problems lasted 20–60 h for most SMP runs, 20–25 days for SMP-DRQN runs and 40–50 days for DRQN runs, on the IRIDIS4 supercomputer (University of Southampton, 2017).

4.2. Learners

To investigate the impact of various learning properties in the mentioned environments, the following learners were implemented in `python` code:

- **SMP**: the above-mentioned implementation of the generic architecture without perception module is used as the baseline SMP. This is the same as Incremental Self-improvement (IS) in Schmidhuber (1999), except the instructions `jump`, effects of which can be achieved using other instructions, and `getP`, an instruction which is rarely included in other experiments.
- **SMP-Fixed**: A perception module is added to the above SMP, to generate an exemplar of active adaptive perception. The perception module is a single feed-forward neural network which outputs external actions whenever the instruction `getOutput` is called, taking as inputs the input cells in the working memory. Thus, the instruction module may generate external actions directly, for example by generating `north`, or indirectly by calling `getOutput`. The network is a fully connected network with two hidden layers of 10 neurons each.
- **SMP-Constructive**: This condition further adds network construction instructions `addNode` and `addConnection` to the SMP-Fixed architecture. Similarly to NEAT, the networks start as a fully connected network without any hidden units.
- **DRQN**: this condition replicates the Deep Recurrent Q-Network with random bootstrapped updates (Hausknecht & Stone, 2015). It was included as an off-policy deep reinforcement learner, using experience replay to more efficiently learn by sampling experiences from an experience buffer and using a target network for improved stability. Two changes were made due to the domain: first, because the observation is small and has no spatial correlations, the convolutional layer was replaced with a dense layer, resulting in a topology of two hidden layers, one dense with 50 RELU-neurons and one LSTM with *tanh*-neurons; second, due to the non-episodic setting, the experience buffer is organised as a single episode rather than a multitude of episodes. To implement DRQN, existing code from VizDoom-Keras-RL, cf. https://github.com/flyyufelix/VizDoom-Keras-RL/blob/master/drqn.py, was modified to the non-episodic setting and to allow the utilisation of the target-network in experience replay.
- **SMP-DRQN**: to provide a second example of the perception module, this learner utilises the same network as the DRQN condition, but enables the SMP to utilise it selectively as a special loop instruction `doQuntil`, with self-chosen exploration rate and self-chosen termination conditions. The DRQN network is modified using `trainReplay` which performs experience replay with a self-chosen batch size while `setExperience` is used to construct a set of useful experiences for finishing `doQuntil`.

Parameter settings are mentioned in Appendix B.

5. Experimental demonstration of active adaptive perception

Behavioural assessment. Choices of the agents were visually inspected on heat-maps with arrows indicating the most frequently chosen action at each position. In the easy mazes, methods using an LSTM network, namely SMP-DRQN and DRQN are able to memorise the path to the goal, while the other SMPs only learn a basic sense of direction. In the difficult mazes, more differences between the learners emerge:

- SMP has a probabilistic preference for single default direction which is best leading to the goal;
- SMP-Fixed and SMP-Constructive briefly check detracting corridors before avoiding them, and frequently visit the best corridors. These methods are not completely able to disambiguate their current state, but rather their networks are similar to a Markovian policy in which faulty choices usually do not lead away from goal, and their $\mathcal{P}$-matrix is similar to the SMP, choosing a single direction;
- Early in the lifetime, DRQN memorises the path towards the goal nearly optimally in 4 out of 10 unique mazes, but gets stuck frequently in the other mazes. The detracting corridors and rooms in those mazes were either greater in number or further from goal. Towards the end, two of those unique mazes keep causing problems with getting stuck. These findings were consistent in the sense that the stuck frequency depended reliably on the maze's topology rather than on network initialisation;
- SMP-DRQN similarly has nearly optimal behaviour on those 4 mazes, and only rarely gets stuck in other mazes. The network's output is similar to DRQN but on detracting corridors, where DRQN fails, the method ignores the network and relies on the $\mathcal{P}$-matrix for a global sense of direction, similar to the SMP;

This illustrates the difficulty of traditional SMPs with perception, the difficulty of deep reinforcement learners in atypical environments, and that active adaptive perception may remedy these problems.

A representative example of the final policy is included for one of the most challenging mazes in Fig. 3, illustrating that methods of active adaptive perception avoid misleading corridors and rooms more often than other methods. Fig. 4 illustrates behaviours observed for SMP-DRQN during the early to middle stages of the lifetime, showing how SMP-DRQN used its perception module less frequently when it was not reliable. Video material[2] shows the behaviours of DRQN and SMP-DRQN on the mentioned example mazes.

Correctness and perception-correctness. The correctness, the proportion of moves that lead the agent closer to the goal, is displayed in Table 2. Methods utilising an LSTM network, DRQN and SMP-DRQN, were characterised by relatively high correctness, and their performance was highly correlated with correctness, indicating their performance is dependent on memorising a correct path. For difficult environments SMP-DRQN did not have a positive correlation, suggesting additional strategies beyond path memorisation. This is in line with the observation that the SMP-DRQN had a performance advantage compared to DRQN in the difficult-random condition, where path memorisation is more challenging. As exemplified in Fig. 4b, it can be observed that the perception-correctness, the correctness of the external actions taken due to the perception module's advise, ignoring external actions directly output by the instruction module, varied strongly over the map. The DRQN system, illustrated in Fig. 4f, had a low correctness in detracting corridors and rooms, and a high correctness close to the goal, and the same finding was observed for the DRQN network when used as the perception module of SMP-DRQN. The explanation for this finding is that initially in the challenging mazes, the system gets stuck for prolonged time in detracting corridors and rooms, without obtaining any rewards. This leads to erroneous and low Q-values for the visited locations on the map. When later the DRQN system more regularly obtains reward, the detracting corridors and rooms maintain such Q-values for a longer time since they usually do not lead to near-term rewards: far from goal, those corridors and rooms have the lowest Q-values; while close to goal they had lower Q-values than locations which were distant but on path to the goal. Later in the lifetime, SMP-DRQN's perception-correctness was higher than DRQN in environments such as those in Fig. 3, where the DRQN got stuck indefinitely. This is because DRQN remains inside a detracting room or corridor and does not reach the reward location, filling the experience buffer with useless experiences. By contrast, SMP-DRQN was able to escape detracting rooms and corridors throughout the lifetime due to the mechanisms of selective network usage and goal-based exploration, and this helped to provide the perception module with useful experiences to learn more efficiently. Therefore, because the experiences are added to the experience buffer at each time step, regardless of whether or not the perception module was used, both mechanisms contribute to an intelligent exploration mechanism. An additional observation in the heatmaps is that when the learner was on the dead-end spaces the DRQN module had low correctness, despite there being only a single action that does not lead to bumping into an obstacle; this occurred either when it was used alone or embedded into SMP-DRQN. This behaviour is due to the combination of two reasons: compared to some other works, for example the T-mazes reported in Bakker (2002), there is no negative reward incurred for bumping into obstacles or any other incorrect actions, and there is no positive reward for correct actions; in addition, the incorrect actions do not lead away from the goal at these locations and therefore these actions only delay the reward achievement by one time step, resulting in smaller differences between the different actions' Q-values; this makes dead-end locations more difficult to learn than other locations for which incorrect actions lead to significant delays in reward achievement. The SMP-DRQN was better able to escape such dead-ends by using a high exploration rate and low network-usage at those locations.

Comparatively, SMP-Fixed and SMP-Constructive have a low correctness and, in difficult environments, the random policy, despite its poor performance, has higher correctness than these two methods. Their perception-correctness was high in strategic locations such as paths leading up to the area with the reward or away from a detracting corridor or room, and incorrect decisions usually are not detrimental to performance as can be observed numerically by the absence of positive correlation between reward speed and correctness. This is related to the visual observations that the decisions made usually did not lead further into wrong corridors, which helps to explain the paradox that although the correctness of SMP-Fixed and SMP-Constructive is low their performance is good. For all conditions, the standard deviation of the correctness over mazes was considerably higher for SMP-Fixed and SMP-Constructive than for other methods. This higher variability may indicate that the learning strategy is more dependent on the features of the environment.

Network nodes and connections. In the network construction of SMP-Constructive a pattern emerged in which the runs with good performance form a greater number of connections, 2000–4000, and maximise the number of nodes n_{nodes} in the network, specifically 176 for easy problems and 276 for difficult problems (cf. Appendix B for parameter settings). The runs with bad performance would end up with a small number of neurons, 20–90, with the difficult-random condition yielding the lowest cumulative reward and the lowest number of nodes, 20–40. This is supported by the correlation between the number of nodes and the reward speed which was medium to high, 0.60–0.93, over the various conditions. However, there were several exceptionally small networks which resulted in excellent performance. For example, in the difficult-fixed condition, a network of 34 neurons resulted in a lifetime average reward of 0.089 on maze 1 which was much larger than for SMP-Fixed, 0.037, and SMP, 0.013. Since SMP-Fixed was able to perform well with just 20 hidden units, this suggests that constructive modifications were only accepted by the evaluation module to the extent other modification types introduced during that modification sequence were useful.

Network usage. The neural network usage, which is the proportion of times the perception module was used to output an external action, developed similarly in SMP-Fixed and SMP-Constructive. It started out small at 20%, but gradually the system started to rely on the network for its instructions, reaching 30% for easy problems and 40% for difficult problems. The discrepancy between easy, 30%, and difficult, 40%, is possibly due to the longer learning time. The heat-maps indicated that the network usage was uniformly spread over the different locations on the map, meaning the learner relied consistently on the perception module. The network usage of SMP-DRQN is higher as advice on several steps are given after a single call of `doQuntil`. In easy environments, SMP-DRQN starts with 5%–20% network usage and develops up to 50%–70%. In difficult environments, eventually the learner relied on the perception module 90% of the time. Unlike the other methods, the network usage of SMP-DRQN was not evenly spread, especially during the early to middle stages of the lifetime: on areas close to the goal, the network

2 https://www.youtube.com/watch?v=xRh-ZXkUJ2Y.

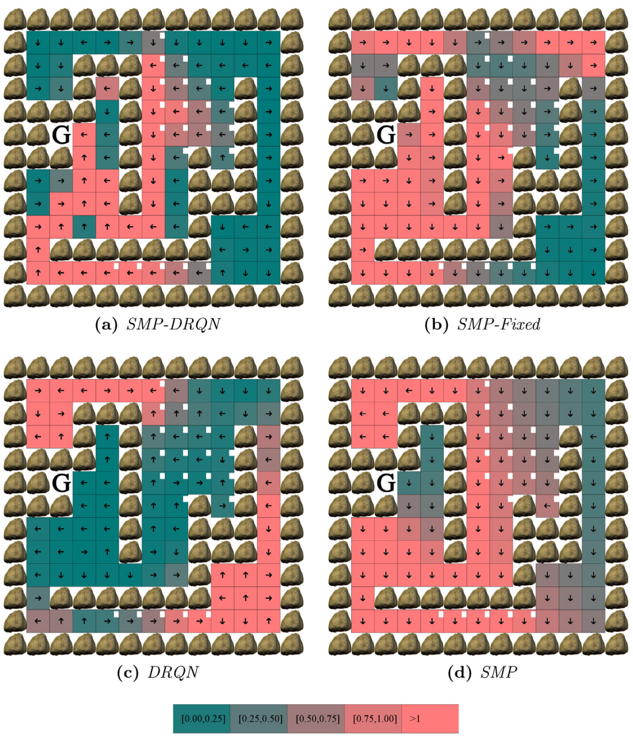


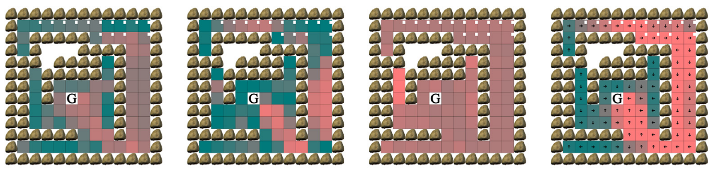
Fig. 3. Heat-map of the final policy on a maze from the difficult-random condition; SMP-Fixed is here taken to represent the first implementation of active adaptive perception since its behaviour is comparable to SMP-Constructive. Though not visible to the agent, the goal location is illustrated by "**G**" while white boxes indicate starting positions. The legend displays the meaning of the colours of the heat-map in terms of visitations per time unit times the number of unique visited locations. $\rightarrow$: arrows indicate the direction of the most frequently chosen action (north, east, south, or west).

Table 2
The correctness metric for the different learners, indicating the proportion of choices made that bring the agent closer to the goal. C_i and C_f are the values of the correctness metric averaged over the various runs during the first and last time slice. The time slice is 1 million time steps for easy and 16 million time steps for difficult. r is the correlation between the lifetime average correctness measure and the lifetime average reward speed.

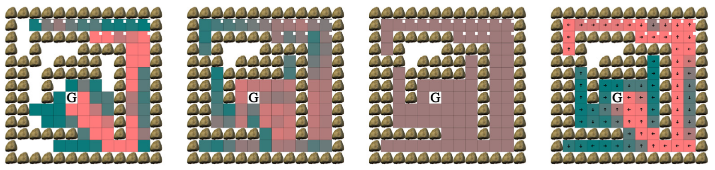
Method	Environment											
	Easy-fixed			Easy-random			Difficult-fixed			Difficult-random		
	C_i	C_f	r	C_i	C_f	r	C_i	C_f	r	C_i	C_f	r
DRQN	.48	.79	.75	.47	.77	.87	.41	.56	.40	.40	.55	.40
Random	.32	.32	−.69	.33	.33	−.74	.36	.36	.16	.36	.36	.14
SMP	.44	.43	.89	.42	.42	.89	.34	.33	−.67	.33	.32	−.74
SMP-Fixed	.41	.38	.54	.41	.42	.62	.32	.32	.26	.30	.30	.18
SMP-Constructive	.40	.37	.15	.39	.38	.47	.31	.29	−.58	.32	.32	−.06
SMP-DRQN	.61	.72	.93	.58	.71	.95	.38	.55	.06	.37	.62	−.27

usage was 70%–90%, whereas on detracting corridors the network usage was between 20 and 60%. Combined with the fact that the network correctness was much lower in those areas, as illustrated in Fig. 4a, this means that SMP-DRQN applied DRQN when it was reliable, such as the paths close to the goal location, but applied a more basic sense of direction where DRQN was not reliable, such as the detracting corridors. This explains why SMP-DRQN performs better in environments where DRQN gets stuck. During the end of the lifetime, the network's correctness in corridors was improved and this resulted in more uniformly high network usage.

Valid modifications. Those modifications maintained at the end of the lifetime, the valid modifications, yield insights into how the agent is learning as they record those changes that accelerated reward intake. These include $\mathcal{P}$-modifications which alter the instruction modules probability matrix and network-modifications which change the network of the perception module. The valid modifications are illustrated in Figs. 6 and 7. In easy problems,



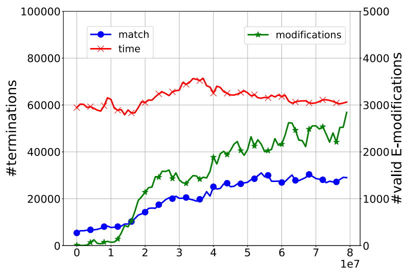
(a) *SMP-DRQN network usage (Pr)* **(b)** *SMP-DRQN Perception-correctness (Pr)* **(c)** *SMP-DRQN Exploration rate (Eps)* **(d)** *SMP-DRQN Heatmap (Vis)*



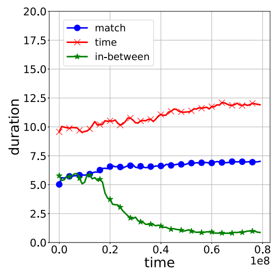
(e) *SMP-DRQN termination experiences (Vis)* **(f)** *DRQN Correctness (Pr)* **(g)** *DRQN Exploration rate (Eps)* **(h)** *DRQN Heatmap (Vis)*

Pr=[0,.20]	Pr=[0.20,0.40]	Pr=[0.40,0.60]	Pr=[0.60,0.80]	Pr=[0.80,1]
Vis=[0,.25]	Vis=[0.25,0.50)	Vis=[0.50,0.75]	Vis=[0.75,1.0]	Vis>1
Eps=[0,.04]	Eps=[0.04,0.08]	Eps=[0.08,0.12]	Eps=[0.12,0.16]	Eps=[0.16,0.20]

Fig. 4. Illustration of the mechanisms behind SMP-DRQN's performance. The first principle is the *selective network usage*: panel (a) shows that the proportion of perception module usage is particularly low in detracting corridors, whilst panel (e) shows that this is due to how the system matches termination experiences on paths to the goal, halting the network usage before reaching detracting corridors; panels (b) and (f) illustrate the perception-correctness of SMP-DRQN and the correctness of DRQN is high on paths close to the goal and highly incorrect far from goal and in detracting corridors; together these illustrate that SMP-DRQN uses its perception module selectively on locations with high perception-correctness, ignoring it when it is not reliable. The second principle is the *goal-based exploration*: panels (c) and (g) illustrate the exploration rate of SMP-DRQN is often higher than DRQN in difficult environments, and especially so on detracting corridors. Together these two principles allow SMP-DRQN to better escape detracting corridors than DRQN, as illustrated in panels (d) and (h). **Note:** the colour of the plots is variable across figures, and their units are mentioned in parentheses; *Pr* is the proportion, *Eps* is the ϵ parameter for the ϵ-greedy action selection, and *Vis* is the visitations per time unit times the number of unique visited locations.



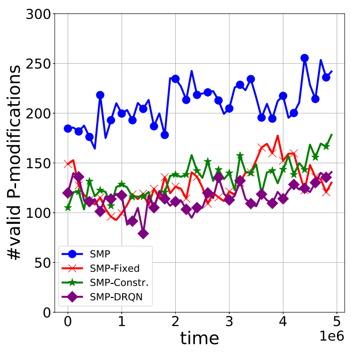
(a) *Match*



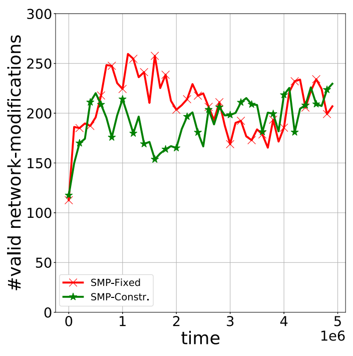
(b) *Duration*

Fig. 5. Illustration of the perception module's goal-matching in the difficult-random condition. Panel (a) illustrates increasing goal-matching: the left *y*-axis illustrates the number of network usage terminations due to **match**, the number of times the system matches the self-chosen termination experience and due to **time**, when the looptime of the `doQuntil` instruction exceeds the self-chosen *until* parameter; the right *y*-axis illustrates the valid number of **modifications** to the experience set E which contains the termination experiences. Panel (b) illustrates how the duration for **match** and **time** increase over time, indicating the learner selects a higher *until* parameter for the `doQuntil` instruction, and how **in-between**, the time in between `doQuntil` loops, decreases over time as the perception-correctness becomes high across the map.

the number of valid modifications is spread evenly across time with the different learners making a similar number of valid modifications. The valid modifications are illustrated for the difficult-random environments in Fig. 7. For all SMPs, a brief initial learning effect is observed, similar to the initial performance gains observed in all learners, since improving on an initial faulty policy is easy. After the initial learning has passed, *learning to learn* is taking place: the learners increasingly learn to generate difficult-to-find modifications that will further accelerate future reward intake. At the end of the lifetime there is a recency



(a) $\mathcal{P}$-*modifications*



(b) *network-modifications*

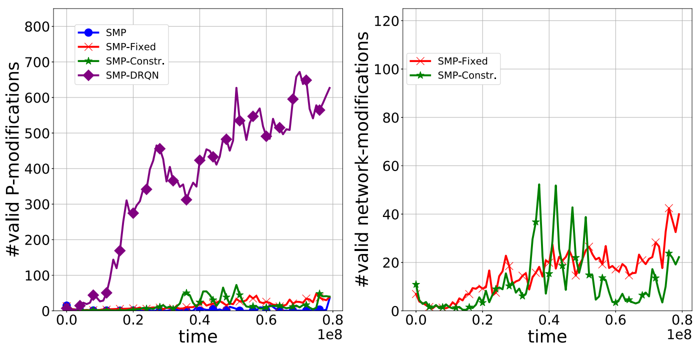
Fig. 6. Development of the valid modifications in the easy-fixed condition. Valid modifications are those changes that were successful according to the Success Story Criterion, indicating lifetime reward acceleration. Each point in the plot thus represents the number of modifications, introduced in a particular time interval $[t, t+\delta]$ with $t \in [0, T)$, which remained in use at the end of the lifetime T, after repeated SSA evaluations.

effect, a sudden peak in valid modifications as a direct result of halting the lifetime at that point: since recent modifications have only been evaluated a few times, the SSA has not yet removed changes which do not accelerate reward in the long run. Compared to the difficult-random condition, the results for the difficult-fixed condition are more monotonously increasing over time but similarly had a brief initial and final peak. A difference between the learners emerges in the second phase where SMP-Fixed and SMP-Constructive have a much greater number of valid $\mathcal{P}$-modifications, with typical peaks of 25–50 and 50–75, respectively, compared to the traditional SMP with peaks of 5–15. For SMP-DRQN there is a continuously increasing curve, eventually reaching a peak of nearly 700 modifications. This higher amount of $\mathcal{P}$-modifications of the active adaptive perception implementations indicates that most of the useful policy changes involve finding out when and how to modify and utilise the neural network perception module. SMP-Fixed and SMP-Constructive also display a similar pattern on the network-modifications, indicating they have learned how to perform useful modifications to the network weights and topology. Other SMP-DRQN development statistics are mentioned in the following subsection.

SMP-DRQN development statistics. The SMP-DRQN system performs two types of perceptual modifications: `trainReplay` and `setExperience`. `trainReplay` is similar to DRQN's usual experience replay and therefore is not proposed to be the main mechanism behind the performance advantage of SMP-DRQN; this is supported by the lower training frequency exhibited by SMP-DRQN. `setExperience` makes changes to the experience set which are later evaluated by SSA. The `setExperience` and `doQuntil` instruction appeared to be key to SMP-DRQN's performance advantage by enabling selective network usage and goal-based exploration. The selective network usage, using DRQN only where it has reliably memorised the path to the termination experience selected by the instruction module, allows the perception module to be used only when the DRQN is advantageous. In areas where DRQN performs poorly the instruction module can directly output external actions. This allows, for example, escaping rooms where the DRQN is stuck. A second factor is goal-based exploration. This allows the instruction module to determine which exploration rates should be chosen together with which termination experiences, meaning that high exploration rates can be set in areas where the learner does not recognise where it is or what is the best action. In the maze tasks, these two factors allow the system to escape detracting corridors and rooms, finding more rewards. Due to reaching the reward location more often initially, these learners can also accumulate more useful experiences compared to learners which get stuck.

The selective network usage is enabled by the instruction module selecting the `doQuntil` instruction and its two key parameters: the *term* experience, an experience taken from the experience set E, and the *until* parameter, a time limit to network usage. The `doQuntil` instruction then repeatedly requests external actions from the perception module until the current experience matches *term* or until the loop time exceeds *until* time steps. The results of this matching process are illustrated in Fig. 4e, where it can be seen that the successfully reached *term* experiences include strategic locations on the path from start to the goal, avoiding usage in detracting corridors. When the *term* experiences are not matched, the network is not used for prolonged amounts of time in detracting corridors and rooms due to the time limit of *until* time steps. Fig. 5a further illustrates that the system was able to better match the self-chosen termination experiences over time. This is not only due to the `setExperience` instruction modifying the experience set E and the increasing network-correctness due to the experience replay, but also, as illustrated in Fig. 5b, due to the $\mathcal{P}$-modifications, which increase the *until* parameter to allow itself more time to reach the more difficult goals. Fig. 5b also illustrates why towards the end of the lifetime, the network usage is uniformly high: as the network's correctness increases, the system learns it can boost the reward speed by increasing the *until* parameter and the frequency of the `doQuntil` instruction.

The goal-based exploration is enabled by the instruction module's choice of the exploration rate, as the third parameter of the `doQuntil` instruction, together with the self-chosen *term* experience which serves as a goal. Illustrative of this principle, the exploration rate was dependent on the difficulty of the environment and the chosen termination experiences: in easy environments rates were lower, with some experiences having an exploration rate between 0.02 and 0.05, most around 0.05–0.11, and the highest average exploration rate is $\epsilon = 0.12$, whereas in difficult environments, most experiences were associated with an exploration rate between 0.09 and 0.12, some were between 0.02 and 0.08, and others between 0.13 and 0.16. This suggests



(a) $\mathcal{P}$-modifications (b) network-modifications

Fig. 7. Development of the valid modifications in the difficult-random condition. Valid modifications are those changes that were successful according to the Success Story Criterion, indicating lifetime reward acceleration. Each point in the plot thus represents the number of modifications, introduced in a particular time interval $[t, t + \delta]$ with $t \in [0, T)$, which remained in use at the end of the lifetime T, after repeated SSA evaluations.

that the system learns which termination experiences are more difficult to achieve and therefore require more exploration. This finding is supported by exploration maps such as those in Fig. 4c, where it can be observed that detracting corridors have relatively high exploration rates compared to DRQN.

Average performance. The development plots in Figs. 8 and 9 display the development of reward speed, the average reward per time step, divided by the optimal reward per time step. On the easy mazes, SMP-DRQN and DRQN obtains the highest reward speeds. DRQN obtains a final reward speed close to 0.70 while SMP-DRQN is just above 0.60. Other SMP conditions are just above 0.30. In the difficult problems SMP-DRQN and DRQN are by far the top performers on the average reward speed. DRQN obtains a final reward speed around 0.4 while SMP-DRQN obtains reward speeds of 0.4 and 0.5 in the fixed and random condition, respectively. Compared to the development in easy problems, more differences emerged between the different SMPs. SMP-Fixed and SMP-Constructive are continuously improving across the lifetime while SMP only initially found good policy improvements. In the fixed condition, this leads to a final reward speed of 0.1 for SMP-Fixed and 0.08 for SMP-Constr, while SMP a speed of 0.025. The random starting position gives a similar performance for SMP, 0.02 across the lifetime, .08 for SMP-Fixed and 0.07 for SMP-Constructive. In difficult problems, it can also be observed that while SMP-Constructive initially learns more quickly, its learning rate slows down compared to SMP-Fixed after around $5 * 10^6$ steps.

As illustrated in Table 3, DRQN obtained the best lifetime average in the easy environments, 0.615 (fixed) and 0.572 (random), but SMP-DRQN obtained the best lifetime average in difficult environments, 0.310 (fixed) and 0.361 (random). Table 3 further shows pair-wise F-tests conducted on the lifetime average reward speed to analyse whether or not between-condition variability was significantly higher than within-condition variability. For the easy problems, no significant effects are found except for the SMP-DRQN and DRQN learners which significantly outperform all other learners. In the difficult problems, the performance of both SMP-Fixed and SMP-Constructive leads to significant effects when compared to SMP. This indicates that rather than maze variability, the principle of active adaptive perception explains why SMP-Fixed and SMP-Constructive outperform SMP. In turn, the difference between SMP-DRQN and DRQN was not significant while pair-wise differences of these learners to SMP-Fixed and SMP-Constructive were significant.

Other performance metrics. The average reward speed, even when normalised, does not necessarily imply superiority, because an excellent relative performance in the most difficult environment will not contribute as much as an excellent absolute performance in a less challenging environment. To resolve this issue, additional metrics illustrate this comparison in Table 4. In the easy mazes, it is clear that DRQN performs the best on all metrics, followed closely by the SMP-DRQN; a more extended lifetime could potentially overcome this given the trend in both development plots. In the difficult mazes, SMP-DRQN has the best average rank, scoring among the top performers consistently, and is followed by DRQN and SMP-Fixed which had the same average rank. The performance ratio, the ratio of the method's average performance to the average performance of the best ranked method, illustrates that SMP-DRQN has the best relative performance, followed by DRQN. Finally, the stuck frequency measures how prone the learner is to get stuck without obtaining rewards; on this metric, the DRQN learner clearly performs worst. To illustrate the statistical significance of the stuck frequencies, pair-wise F-tests comparing the SMPs to the DRQN learner yielded $p = 0.092$ for SMP, $p = 0.036$ for SMP-Fixed, $p = 0.065$ for SMP-Constructive and $p = 0.094$ for SMP-DRQN in the difficult-fixed condition, and $p = 0.063$ for SMP, $p = 0.035$ for SMP-Fixed, and $p = 0.034$ for SMP-Constructive and $p = 0.033$ for SMP-DRQN in the difficult-random condition. This means that based on a threshold for significance $\alpha = .05$, all learners with active adaptive perception had significantly lower stuck frequency in the difficult-random condition, supporting observations that they avoided detracting corridors and rooms more easily.

6. Discussion

Similar to earlier SMPs, the proposed architecture is coordinated by a self-modifying policy which interacts with itself and other functional components by means of instructions. This is a general approach due to the way in which any sort of instruction may be utilised for learning how to maximise the cumulative reward. The results of the experiments have provided evidence that the addition of active adaptive perception as an additional mechanism in SMPs makes the architecture more suitable

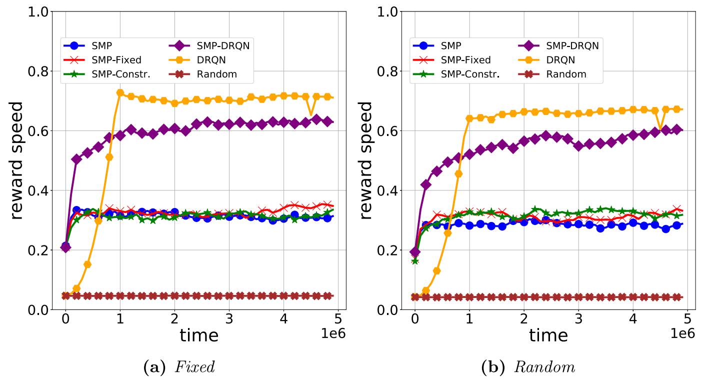


Fig. 8. Development plots of the reward speed for the easy-fixed and the easy-random condition, over the lifetime of 5 million time steps. For each plot reward speed, the average reward per time step, is averaged over 20 runs, 2 repetitions for each of the 10 mazes, and normalised in [0, 1] such that the optimal speed gives performance of 1.0.

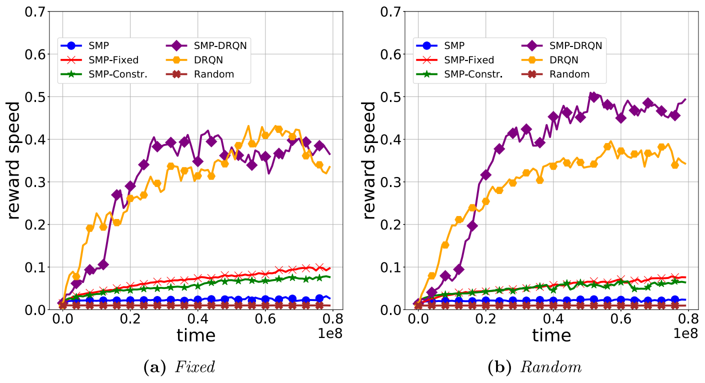


Fig. 9. Development plots of the reward speed for the difficult-fixed and difficult-random condition, over the lifetime of 80 million time steps. For each plot reward speed, the average reward per time step, is averaged over 20 runs, 2 repetitions for each of the 10 mazes, and normalised in [0, 1] such that the optimal speed gives performance of 1.0.

for recognising complex situations and constructing perceptual learning strategies. A first implementation with a feedforward network using computationally cheap instructions and trained without an explicit loss function was implemented to illustrate emergence of simple strategies: rather than learning correct responses across the map, the learners discovered how to adjust the neural network to maintain only those modifications that lead to lifelong reward acceleration, by focusing on those areas where learning yields the most benefits. In the maze experiments this manifested itself by the selective optimisation of the network for particular parts of the map. As the neural network was adapted in this way, it was used more often as time went by. The fact that even simple instructions allowed consistent learning in difficult environments is a strong statement, since more efficient instructions are likely to yield greater benefits. A second implementation utilised a recurrent Q-network selectively where it is reliable and the SMP's probabilistic sense of direction otherwise, allowing direct performance benefits but also indirect benefits by providing useful experiences to improve the Q-network's accuracy. The learner was also able to select the exploration rate for epsilon-greedy action selection dependent on its self-chosen goals and found that difficult goals require higher exploration rates. Earlier studies with SSA (Schmidhuber, 1999; Schmidhuber, Zhao and Wiering , 1997) have demonstrated the ability to optimise the real-time performance. Because the SSA is part of the active adaptive perception implementation, it is expected that the current implementation is suitable for real-time environments. Additionally the proposed architecture is a novel method for composing training and construction algorithms for neural networks, it can evolve a network in non-episodic environments, unlike Topology and Weight Evolving Artificial Neural Networks such as NEAT (Stanley & Miikkulainen, 2002), and compared to Constructive Neural networks (Fahlman & Lebiere, 1990; Fanguy & Kubat, 2002; Frean, 1990; Lahnajarvi, Lehtokangas, & Saarinen, 2002; Parekh, Yang, & Honavar, 2000; Ring, 1997; Sharma & Chandra, 2010; Vamplew & Ollington, 2005) the architecture

Table 3
Life-time averaged normalised reward speed. $>$ and $<$ are used to indicate whether the method's performance is higher or lower than its comparison, and p denotes the significance value of the pair-wise F-test.

	Method	Performance	Comparison				
			DRQN	SMP-Constr.	SMP-Fixed	SMP	Random
Easy-fixed	SMP-DRQN	0.592 ± 0.132	$< p = 0.644$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.615 ± 0.069	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.312 ± 0.120	/	/	$< p = 0.807$	$< p = 0.982$	$> p < 0.001$
	SMP-Fixed	0.325 ± 0.111	/	/	/	$> p = 0.813$	$> p < 0.001$
	SMP	0.313 ± 0.106	/	/	/	/	$> p < 0.001$
	Random	0.046 ± 0.019	/	/	/	/	/
Easy-random	SMP-DRQN	0.542 ± 0.164	$< p = 0.649$	$> p = 0.003$	$> p = 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.572 ± 0.113	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.314 ± 0.142	/	/	$> p = 0.923$	$> p = 0.611$	$> p < 0.001$
	SMP-Fixed	0.308 ± 0.124	/	/	/	$> p = 0.657$	$> p < 0.001$
	SMP	0.284 ± 0.118	/	/	/	/	$> p < 0.001$
	Random	0.042 ± 0.019	/	/	/	/	/
Difficult-fixed	SMP-DRQN	0.310 ± 0.109	$> p = 0.909$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.304 ± 0.135	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	0.056 ± 0.027	/	/	$< p = 0.425$	$> p = 0.001$	$> p < 0.001$
	SMP-Fixed	0.069 ± 0.041	/	/	/	$> p = 0.002$	$> p < 0.001$
	SMP	0.023 ± 0.013	/	/	/	/	$> p < 0.001$
	Random	0.010 ± 0.004	/	/	/	/	/
Difficult-random	SMP-DRQN	0.361 ± 0.075	$> p = 0.294$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	DRQN	0.295 ± 0.176	/	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$	$> p < 0.001$
	SMP-Constr.	$0.050 + 0.030$	/	/	$> p = 0.765$	$> p = 0.011$	$> p < 0.001$
	SMP-Fixed	0.054 ± 0.029	/	/	/	$> p = 0.003$	$> p < 0.001$
	SMP	0.021 ± 0.013	/	/	/	/	$> p = 0.008$
	Random	0.010 ± 0.004	/	/	/	/	/

Table 4
Additional performance metrics, illustrated with the average and standard-deviation across runs, for the different conditions (a) easy, and (b) difficult. **rank** indicates the rank, ranging between 1.0, always best, and 6.0, always worst. **ratio** indicates the ratio of performance to the performance of the best of both, yielding 1 if it is the best, otherwise a number in [0, 1). **stuck** is the proportion of consequent samples in which the cumulative reward did not increase, with a sampling rate of once every 10 000 time steps.

	Fixed			Random		
	rank	ratio	stuck	rank	ratio	stuck
(a) Easy						
DRQN	1.2 ± 0.4	$.97 \pm .08$	$.00 \pm .00$	1.4 ± 0.5	$.97 \pm .06$	$.00 \pm .00$
Random	6.0 ± 0.0	$.07 \pm .03$	$.00 \pm .00$	6.0 ± 0.0	$.07 \pm .03$	$.00 \pm .00$
SMP	3.9 ± 0.8	$.49 \pm .16$	$.00 \pm .00$	4.3 ± 0.8	$.46 \pm .16$	$.00 \pm .00$
SMP-Fixed	$3.9 + 0.8$	$.50 \pm .16$	$.00 \pm .00$	3.7 ± 0.8	$.51 \pm .18$	$.00 \pm .00$
SMP-Constructive	4.2 ± 0.8	$.48 \pm .16$	$.00 \pm .00$	3.8 ± 0.8	$.52 \pm .21$	$.00 \pm .00$
SMP-DRQN	$1.7 + 0.4$	$.91 \pm .12$	$.00 \pm .00$	1.7 ± 0.7	$.90 \pm .14$	$.00 \pm .00$
(b) Difficult						
DRQN	1.9 ± 1.4	$.63 \pm .43$	$.20 \pm .25$	2.3 ± 1.9	$.74 \pm .39$	$.24 \pm .29$
Random	5.8 ± 0.5	$.04 \pm .07$	$.01 \pm .01$	5.7 ± 0.4	$.05 \pm .03$	$.01 \pm .01$
SMP	4.8 ± 0.6	$.08 \pm .09$	$.04 \pm .12$	4.9 ± 0.5	$.06 \pm .03$	$.05 \pm .10$
SMP-Fixed	3.2 ± 0.5	$.22 \pm .16$	$.02 \pm .03$	3.2 ± 0.5	$.14 \pm .08$	$.03 \pm .06$
SMP-Constructive	3.6 ± 0.9	$.20 \pm .20$	$.04 \pm .07$	3.3 ± 0.7	$.13 \pm .08$	$.04 \pm .06$
SMP-DRQN	1.7 ± 0.9	$.83 \pm .23$	$.05 \pm .05$	1.5 ± 0.5	$.93 \pm .09$	$.03 \pm .02$

does not need heuristic criteria for updating and is suitable for reinforcement learning. Lastly, the architecture for active adaptive perception has demonstrated features typically associated with continual learning, particularly (a) the ability to learn in a single lifetime with no known terminal states, and (b) the ability to learn how to learn incrementally. Although the current experiments have not provided evidence for the ability to learn multiple tasks, the extension to continual and lifelong learning is feasible since an earlier SMP study, utilising the Incremental Self-improvement which serves as the basis for the current implementation, demonstrated the ability to solve different problems of increasing complexity using inductive transfer (Schmidhuber, Zhao and Wiering , 1997).

Comparatively, adding adaptive perception as an additional mechanism in SMPs yielded a continuous learning curve and significant performance gains. For a traditional SMP, it was difficult to find valid self-modifications relating to instructions that did not help the learner perceive the environment, and although it had an overall sense of direction, its working memory operations did not enable it to develop a search strategy. Compared to active perception instructions used in earlier SMPs (Schmidhuber, Zhao and Schraudolph , 1997), the perception module similarly influences the instruction generation but allows long-term adaptation and does not rely on knowledge of the environment.

The deep reinforcement learners included in the study use an LSTM network to learn an action value-function (Sutton & Barto, 2018), an estimate of the discounted cumulative reward for a given history and a given action. These learners were able to memorise the sequence from start to goal when path lengths were short, but did not perform so well when paths were longer and when there were more detracting corridors and rooms. Providing the action-value as the target for backpropagation-through-time is problematic in complex continuing environments. This is because it makes the limited trace length of back-propagation and the discounted cumulative reward imply events in the distant future do not affect action selection. Similarly, although LSTMs do not tend to suffer from the vanishing gradient compared to traditional recurrent neural networks (Bengio, Simard, & Frasconi, 1994; Hochreiter & Schmidhuber, 1997), comparable issues may occur due to the exploding gradient (Sutskever, Vinyals, & Le, 2014). The proposed implementations of active adaptive perception avoid these problems by either not requiring a target, or by selectively applying the action-value network only when it is correct. Even though the DRQN system is normally trained on episodic environments, still this system was successful in the non-episodic environments with sparse feedback. This is attributed to two factors: first, the stability of the Q-function is increased by only updating the target network periodically to the parameters of the model-network which is updated during training; second, the problem of learning even when there is a low diversity of experiences, such as when stuck in a detracting corridor, is addressed by sampling from the experience buffer which stores experiences over a long period of time. Despite this, in the most challenging mazes DRQN gets stuck in detracting corridors and rooms for extended periods of time. SMP-DRQN did not suffer from this problem which is attributed to (a) the ability to ignore the network when it is not reliable; (b) SSA evaluating the agent in the long-term; and (c) goal-based exploration allowing to set the exploration rate depending on a particular target experience chosen by the learner. Although SMP-DRQN overcame some of the issues, several strategies used in deep reinforcement learning are useful for the non-episodic scenario with limited knowledge and sparse feedback: prioritised experience replay (Schaul, Quan, Antonoglou, & Silver, 2016) may focus training on the most problematic experiences; exploration may be stimulated by intrinsic motivation (Singh et al., 2004) or exploration bonuses (Bellemare et al., 2016); average reward reinforcement learning (Mahadevan, 1996; Yang, Gao, An, Wang, & Chen, 2016) may be used to avoid the problems with discounting the future experiences; for decorrelating experiences, asynchronous methods (Mnih et al., 2016) may be used as an alternative to experience replay which is suitable for both off- and on-policy methods and which may make use of parallelism for improved real-time performance.

There are some limitations for the current exemplar method including its reduced relative performance on simple environments, likely because the universality of the method implies that it takes longer to find narrow behaviours from the larger behavioural repertoire. Due to modifying the perception module one parameter at a time, the NEAT implementation is not suitable for large-scale experiments. Larger scale experimentation, whilst maintaining the incremental network parameter updates with long-term evaluations, would be possible by utilising instructions which modify a functional, abstract representation of a network rather than a network itself, similar to HyperNEAT. Moreover, its random increments to the network parameters could be improved: a straightforward extension to the NEAT implementation could be to, in addition to learning which parameters are in need of update, also learn how to increment or decrement the parameters by including the increment as an additional argument to the instruction. The feedforward structure did not solve partial observability, despite including historical variables, and instead additional working cells as inputs or a recurrent structure should be considered. In addition, despite often introducing more complexity, the performance of the constructive network was comparable to a network with fixed topology. One reason may simply be due to the nature of constructive neural networks which tend to learn fast initially but resulted in a similar even sometimes lower final performance due to overfitting on the initially small network (Franco & Conde, 2008; Junior, Nicoletti, & Lu, 2016). In addition, the observed relation between reward intake and addition of nodes and connections suggests that SSA is not noticing small negative effects of constructive changes that go together with large positive effects of weight and instruction probability changes, due to the evaluation of modification sequences rather than individual modifications. The SMP-DRQN implementation addresses some of the above issues, particularly the efficient use of experience, state disambiguation, and the selective application of perception. A limitation of the evaluation module implementation, SSA, is that it uses a stack which can in principle grow indefinitely. While compression of stack entries can reduce memory in practice, this limitation highlights the need for practical long-term evaluation of self-modifying reinforcement learning policies. Also the current system is limited in predictive capabilities: the trial-and-error self-modification yields many unsuccessful self-modifications and there is no extensive world model.

The key conjecture of this paper is that active adaptive perception is preserved even when the implementation is changed significantly; different implementations may be used for each of its four components, as long as the functional requirements are satisfied. For example, the evaluation module need not be the Success Story Algorithm. The perception module may consist of not one but several sub-symbolic components such as neural networks, support vector machines or clustering methods, and perceptual advice does not necessarily output external actions but may be any operation which temporarily influences instruction generation. For example, it may make temporary changes in the probabilities of neighbouring cells or change the contents of internal variables to generate instructions based on a classification of the agent's state. The instruction module may generate its programmatic instructions using a representation different than a probability matrix. The working memory could be implemented differently to use real numbers instead. Similarly, the interactions between the components may be directed by a different set of instructions. Additional components suitable for cognitive architectures may be added for further gains in complex tasks.

7. Conclusions

To address the need for universal reinforcement learners, this paper investigated how a self-modifying reinforcement learning policy may benefit from active adaptive perception, the ability to modify and utilise perceptual modules in completely self-chosen ways. This ability enables a learner to invent strategies for discriminating various situations to help achieve goals in complex environments. It does this by learning how to modify its own learning operations based on incoming rewards. As an illustration, two exemplar systems with active adaptive perception were compared to other methods on non-episodic partially observable mazes with sparse reward structures. The first exemplar learned to modify and use a feedforward network with a simple instruction set based on long-term reward intake of the self-modifying policy, instead of traditionally training the network on an explicit loss function. This leads to simple strategies to avoid detracting corridors and rooms, comparing favourably over a traditional self-modifying policy. A second exemplar system was more computationally expensive, using a recurrent network and experience replay. This system used instructions to determine when and how to apply and update a DRQN network. It learned to selectively apply the DRQN where it was reliable and to select the exploration factor depending on its current goal. This was beneficial compared to DRQN on the most difficult problems where DRQN got stuck in detracting corridors and rooms. The architecture also constitutes a novel framework for training and constructing neural networks by learning to use elementary user-defined instructions.

Acknowledgements

This research was funded by Engineering and Physical Sciences Research Council and LLoyd's Register Foundation. The authors acknowledge the use of the IRIDIS High Performance Computing Facility and associated support services at the University of Southampton.

Appendix A. Maze generation

Mazes were created according to algorithm below. In the fixed condition, the starting position S and G are selected manually such that the distance $d(S,G)$ is in the desired distance range, 4–8 steps for easy vs. 11–30 for difficult mazes. In the random condition, the goal location G is the same and a starting point S is sampled from the set of open spaces with a distance in $[\alpha * d(S, G), \beta * d(S, G)]$. Due to the restricted set of locations in easy problems, setting $\alpha = .5$ and $\beta = 1.2$ for easy and $\alpha = .90, \beta = 1.10$ for difficult resulted in a sufficient number of starting locations.

Algorithm 2 Procedure for generating mazes. Note: / is the integer division.

```
sizeX ← 13; sizeY = 13;
compl ← .10; density ← .10;
length ← floor(complexity ∗ (5 ∗ (sizeX + sizeY)));
islands ← floor(density ∗ ((sizeX/2) ∗ (sizeY/2)));
Fill borders with obstacles;
for i ← 0 to islands − 1 do
    (y, x) ← get-random-position();
    set obstacle on (y,x);
    for j ← 0 to length − 1 do
        neighbours ← ∅;
        if x > 1 then
            neighbours.append((y, x − 2);
        if x < sizeX − 2 then
            neighbours.append((y, x + 2));
        if y > 1 then
            neighbours.append((y − 2, x));
        if  y < sizeY − 2 then
            neighbours.append((y + 2, x));
        if  length(neighbours) > 0 then
            ỹ, x̃ ← random-neighbour();
            if (ỹ, x̃) is free then
                set obstacle on (ỹ, x̃);
                ȳ = ỹ + (y − ỹ)/2;
                x̄ = x̃ + (x − x̃)/2);
                set obstacle on (ȳ, x̄);
                (y, x) ← (ỹ, x̃)
Pick start S and goal G manually.
if condition=Random then
    Initialise α < 1, β > 1
    reachable ← reachable-from(G);
    d_ref ← dist(S,G);
    starts ← ∅
    for p ∈ reachable do
        if dist(p, G) ∈ [αd_ref, βd_ref] then
            starts.append(p);
```

Appendix B. Parameter settings

For the SMPs using Incremental Self-improvement, the number of program cells m was set to 50 for easy and 100 for difficult problems. A minimal probability $minP = .0005$ ensured all instructions were regularly computed. The total number of working memory cells N_{wm}, including input, working, register and output cells, was 130 for easy and 220 for difficult problems. A small change was made to IS to encourage learning over the lifetime, namely, duplicates of each of the self-modification instructions `incP` and `decP` were added to the instruction set $\mathcal{A}$. For the SMP condition, both were duplicated 10 times, yielding $|\mathcal{A}| = 39$ and 22 modification instructions; For the SMP-Fixed condition, both were duplicated 9 times resulting in $|\mathcal{A}| = 39$ instructions, 21 of which were modification instructions; For the SMP-Constructive, both were duplicated 8 times resulting in $|\mathcal{A}| = 39$ instructions, 21 of which were modification instructions; For the SMP-DRQN, both were duplicated 9 times resulting in $|\mathcal{A}| = 40$, 22 of which were modification instructions. The effect of duplication is discussed in Appendix C. In SMP-Constructive, addition of nodes was limited to a maximum of $max = 2MaxInt$ neurons, leading to 176 for easy and 276 for difficult problems. Using the above information, the following parameters were set to determine

Table 5
DRQN parameters.

Parameter	Setting
Unroll	25 (easy), 40 (difficult)
Batch size	32
Replay memory size	400 000 experiences
Initial exploration rate	1.0
Final exploration rate	0.1
Exploration frame	1000 000 time steps
Optimisation algorithm	AdaDelta (Zeiler, 2012)
Learning rate	0.1
Momentum	0.95
Clip gradient	Absolute value exceeding 10
Replay start	50 000 time steps
Update frequency	4 time steps
Target update frequency	10 000 time steps

the addresses of the working memory: $Max = \mathcal{A} + m$; $Min = Max - N_{wm}$; $RegisterStart = 0$; $InputEnd = Min + 8$. The input cells had addresses in $Min, \ldots, InputEnd$, the working cells had addresses in $InputEnd + 1, \ldots, RegisterStart - 1$, the register cells had addresses in $RegisterStart, \ldots, |\mathcal{A}| - 1$, and the output cells had addresses in $|\mathcal{A}, Max|$. The range of representable numbers $[-MaxInt, MaxInt]$ was set using $MaxInt = \max(|Min|, Max)$ where Min is the lowest address and Max is the highest address in the working memory.

For DRQN, all parameter settings, mentioned in Table 5, were the same as in Hausknecht and Stone (2015), except the *unroll* parameter, the trace-length for prediction and backpropagation through time, was set to 25 and 40 for easy and difficult mazes, respectively. The only exception is that the exploration frame of 10^6 time steps used in DRQN, in which the exploration rate is decreased linearly from $\epsilon = 1$ to $\epsilon = .10$, is not required for SMP-DRQN, since (a) the SMP does not necessarily rely on the Q-network; and (b) the exploration rate is controlled by the instruction module via arguments to `doQuntil`. The batch size and exploration rate were adapted dynamically by SMP-DRQN, starting initially from a uniform distribution with the same average as the original DRQN setting, namely, batchsize = 32 and $\epsilon = .10$.

Appendix C. Effect of duplication

Duplication of `incP` and `decP` is suggested to improve the performance of Incremental Self-improvement. Comparative results with and without duplication are shown in Table 6. They illustrate that, even without duplication, the active adaptive perception

Table 6
Effect of duplication of the `incP` and `decP` instructions on the lifetime average of the normalised reward speed. Bold font is used to illustrate the best-performing learner without duplication.

Condition	Learner	Dupl	No Dupl
Easy-fixed	SMP	.313	**.362**[a]
	SMP-Fixed	.325	.323
	SMP-Constructive	.312	.306
Easy-random	SMP	.284	.257
	SMP-Fixed	.308	.258
	SMP-Constructive	.314[a]	**.272**
Difficult-fixed	SMP	.023	.023
	SMP-Fixed	.069[a]	.041
	SMP-Constructive	.056	**.048**
Difficult-random	SMP	.021	.018
	SMP-Fixed	.054[a]	.039
	SMP-Constructive	.050	**.051**

[a]Indicates the best-performing learner in general, duplication or no duplication.

methods always outperform the traditional SMP in the difficult environments. However, duplication provides an additional positive effect on performance. This positive effect is attributed to a greater flexibility in change sizes. For example, if a particular entry has .001 as a probability, performing decP on this entry shrinks the entry to between .00001 and .00099, whereas performing decP on another entry with a probability .10 will decrease its probability to between .001 and .099, which is a much larger absolute decrease. Moreover, there are favourable side-effects: (a) there is a higher initial probability of self-modification instructions since they have multiple entries in the matrix; (b) the combined minimal probability of self-modification is higher since each entry in the probability matrix must have a probability greater than *minP*; (c) an enhanced syntactical correctness, resulting in more correct executions of the self-modification instructions.

References

Adams, S., Arel, I., Bach, J., Coop, R., Furlan, R., Goertzel, B., et al. (2012). Mapping the landscape of human-level artificial general intelligence. *AI Magazine*, *33*(1), 25–42. http://dx.doi.org/10.1609/aimag.v33i1.2322, URL http://www.aaai.org/ojs/index.php/aimagazine/article/view/2322.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060. http://dx.doi.org/10.1037/0033-295X.111.4.1036.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine, Special Issue on Deep Learning for image understanding*, *34*, 26–38, arXiv:1708.05866v2.

Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., et al. (2009). Cognitive developmental robotics: a survey. *IEEE Transactions on Autonomous Mental Development*, *1*(1), 12–34. http://dx.doi.org/10.1109/TAMD.2009.2021702.

Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, *76*(8), 31–37. http://dx.doi.org/10.1016/B978-008045046-9.01436-4.

Bajcsy, R., Aloimonos, Y., & Tsotsos, J. K. (2017). Revisiting active perception. *Autonomous Robots*, (January), http://dx.doi.org/10.1007/s10514-017-9615-3.

Bakker, B. (2002). Reinforcement learning with long short-term memory. In *Advances in neural information processing systems 12 (NIPS 2002)*.

Baum, S. D., Goertzel, B., & Goertzel, T. G. (2011). How long until human-level ai ? results fom an expert assessment. *Technological Forecasting & Social Change*, *78*(1), 185–195.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., et al. (2016). Unifying count-based exploration and intrinsic motivation. In *30th conference on neural information processing systems (NIPS 2016)*. Barcelona, Spain: http://dx.doi.org/10.2172/1336367, URL http://arxiv.org/abs/1606.01868 arXiv:1606.01868.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th international conference on machine learning*. Montreal, Canada.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Cassimatis, N. L. (2002). Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes.

Cassimatis, N. L., Trafton, J. G., Bugajska, M. D., & Schultz, A. C. (2004). Integrating cognition, perception and action through mental simulation in robots. *Robotics and Autonomous Systems*, *49*(1-2 SPEC. ISS.), 13–23. http://dx.doi.org/10.1016/j.robot.2004.07.014.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS cognitive architecture. *Cognitive Systems Research*, *48*, 25–38. http://dx.doi.org/10.1016/j.cogsys.2017.05.005.

Crook, P. A. (2006). *Learning in a State of Confusion: employing active perception and reinforcement learning in partially observable worlds* (Phd thesis), University of Edinburgh.

Everitt, T., Filan, D., Daswani, M., & Hutter, M. (2016). Self-modification of policy and utility function in rational agents. In *The 8th conference on artificial general intelligence (AGI-2016)*. arXiv:1605.03142v1.

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In *Advances in neural information processing systems (NIPS 90)* (pp. 524–532).

Fanguy, R., & Kubat, M. (2002). Modifying upstart for use in multiclass numerical domains. In S. Haller, & G. Simmons (Eds.), *Proceedings of the fifteenth international florida artificial intelligence research society conference (FLAIRS-02)* (pp. 339–343).

Franco, L., & Conde, I. M. (2008). Active learning using a constructive neural network algorithm. In V. Kurkova, R. Neruda, & J. Koutnik (Eds.), *ICANN 2008, Part II, lecture notes in computer science 5164* (pp. 803–811). Prague, Czech Republic: Springer-Verlag Berlin Heidelberg.

Franklin, S., Madl, T., Mello, S. D., & Snaider, J. (2014). LIda : a systems-level architecture for cognition , emotion , and learning. *IEEE Transactions on Autonomous Mental Development*, *6*(1), 19–41.

Frean, M. (1990). The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, *2*(2), 198–209. http://dx.doi.org/10.1162/neco.1990.2.2.198.

Gibson, J. J. (1966). *The senses considered as perceptual systems*. Oxford, England: Houghton Mifflin.

Goertzel, B., Ke, S., Lian, R., Neill, J. O., Sadeghi, K., Wang, D., et al. (2013). The cogprime architecture for embodied artificial general intelligence. In *2013 IEEE symposium on computational intelligence for human-like intelligence (CIHLI)* (pp. 60–67).

Goertzel, B., Pitt, J., Wigmore, J., Geisweiller, N., Cai, Z., Lian, R., et al. (2011). Cognitive synergy between procedural and declarative learning in the control of animated and robotic agents using the opencogprime agi architecture. In *Proceedings of the Twenty-Fifth AAAI conference on artificial intelligence* (pp. 1436–1441).

Gosavi, A. (2009). Reinforcement learning: a tutorial survey and recent advances. *INFORMS Journal on Computing*, *21*(2), 178–192. http://dx.doi.org/10.1287/ijoc.1080.0305.

Hausknecht, M., & Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. In *AAAI fall symposium series* (pp. 29–37). AAAI, arXiv:1507.06527.

Hawkins, J. (2004). *On intelligence*. Times Books, URL www.onintelligence.com.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1–32. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hutter, M. (2007). Universal algorithmic intelligence: a mathematical top-down approach. In *Artificial general intelligence*.

Junior, B., Nicoletti, C., & Lu, R. W. (2016). Enhancing constructive neural network performance using functionally expanded input data. *Journal of Artificial Intelligence and Soft Computing Research*, *6*(2), 119–131. http://dx.doi.org/10.1515/jaiscr-2016-0010.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101* (1–2), 99–134. http://dx.doi.org/10.1016/S0004-3702(98)00023-X, URL http://linkinghub.elsevier.com/retrieve/pii/S000437029800023X arXiv:/doi.org/10.1016/S0004-3702(98)00023-X.

Karnowski, T. P., Arel, I., & Rose, D. (2010). Deep spatiotemporal feature learning with application to image classification. In *2010 ninth international conference on machine learning and applications* (pp. 883–888). http://dx.doi.org/10.1109/ICMLA.2010.138, URL http://ieeexplore.ieee.org/document/5708961/.

Kieras, D. E., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, *12*, 391–438.

Lahnajarvi, J. J. T., Lehtokangas, M. I., & Saarinen, J. P. P. (2002). Evaluation of constructive neural networks with cascaded architectures. *Neurocomputing*, *48*, 573–607.

Laird, J. E. (2012). *The Soar cognitive architecture*. The MIT Press.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, http://dx.doi.org/10.1038/nature14539.

Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., & Shepherd, M. (1990). Cyc: toward programs with common sense. *Communications of the ACM*, *33*.

Li, Y. (2017). Deep reinforcement learning: an overview. In *Proceedings of SAI intelligent systems conference (intellisys) 2016* (pp. 1–70). Springer, http://dx.doi.org/10.1007/978-3-319-56991-8_32, arXiv:1701.07274.

Lin, L. -J., & Mitchell, T. M. (1993). Reinforcement learning with hidden states. In J. -A. Meyer, H. L. Roitblat, & S. W. Wilson (Eds.), *From animals to animats 2* (pp. 271–280). MIT Press.

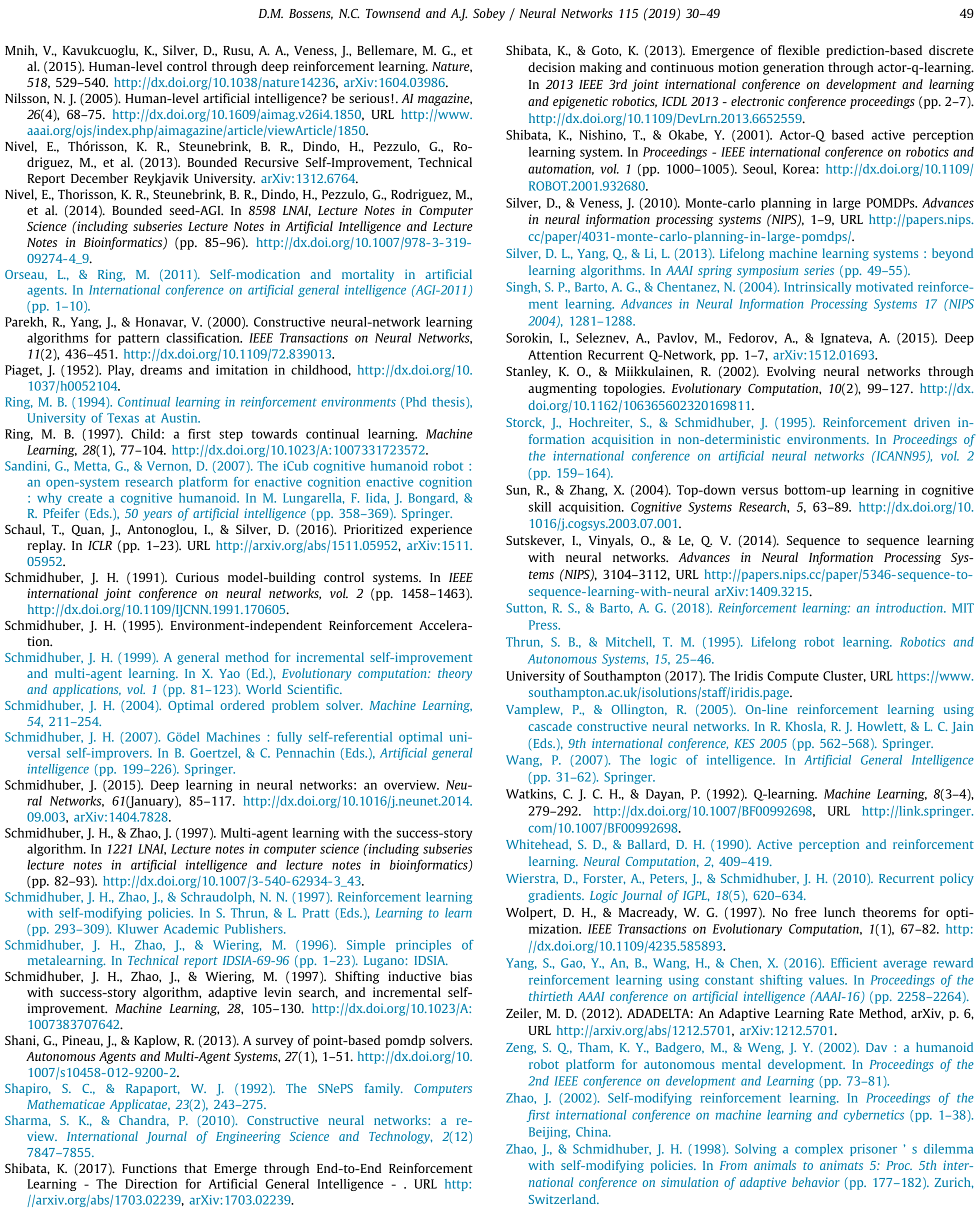
Mahadevan, S. (1996). Average reward reinforcement learning: foundation, algorithms, and empirical results. *Machine Learning*, *22*(1/2/3), 159–196. http://dx.doi.org/10.1023/A:1018064306595, URL http://link.springer.com/10.1023/A:1018064306595.

McCarthy, J. (2007). From here to human-level AI. *Artificial Intelligence*, *171*(18), 1174–1182. http://dx.doi.org/10.1016/j.artint.2007.10.009.

Mitchell, T. M. (1980). The need for biases in learning generalizations. In *Readings in machine learning* (pp. 184–191). Morgan Kaufmann Publishers, URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.5466.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd international conference on machine learning, vol. 48*. New York, NY, USA: http://dx.doi.org/10.1177/0956797613514093, URL http://arxiv.org/abs/1602.01783, arXiv:1602.01783.

Mnih, V., Hess, N., Graves, A., & Kavukcuoglu, K. (2014). Recurrent models of visual attention. In *Proceedings of the 27th international conference on neural information processing systems, vol. 2* (pp. 2204–2212). http://dx.doi.org/10.1017/S037346330300239X, arXiv:1406.6247v1.



Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–540. http://dx.doi.org/10.1038/nature14236, arXiv:1604.03986.

Nilsson, N. J. (2005). Human-level artificial intelligence? be serious!. *AI magazine*, *26*(4), 68–75. http://dx.doi.org/10.1609/aimag.v26i4.1850, URL http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1850.

Nivel, E., Thórisson, K. R., Steunebrink, B. R., Dindo, H., Pezzulo, G., Rodriguez, M., et al. (2013). Bounded Recursive Self-Improvement, Technical Report December Reykjavik University. arXiv:1312.6764.

Nivel, E., Thorisson, K. R., Steunebrink, B. R., Dindo, H., Pezzulo, G., Rodriguez, M., et al. (2014). Bounded seed-AGI. In *8598 LNAI, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (pp. 85–96). http://dx.doi.org/10.1007/978-3-319-09274-4_9.

Orseau, L., & Ring, M. (2011). Self-modication and mortality in artificial agents. In *International conference on artificial general intelligence (AGI-2011)* (pp. 1–10).

Parekh, R., Yang, J., & Honavar, V. (2000). Constructive neural-network learning algorithms for pattern classification. *IEEE Transactions on Neural Networks*, *11*(2), 436–451. http://dx.doi.org/10.1109/72.839013.

Piaget, J. (1952). Play, dreams and imitation in childhood, http://dx.doi.org/10.1037/h0052104.

Ring, M. B. (1994). *Continual learning in reinforcement environments* (Phd thesis), University of Texas at Austin.

Ring, M. B. (1997). Child: a first step towards continual learning. *Machine Learning*, *28*(1), 77–104. http://dx.doi.org/10.1023/A:1007331723572.

Sandini, G., Metta, G., & Vernon, D. (2007). The iCub cognitive humanoid robot : an open-system research platform for enactive cognition enactive cognition : why create a cognitive humanoid. In M. Lungarella, F. Iida, J. Bongard, & R. Pfeifer (Eds.), *50 years of artificial intelligence* (pp. 358–369). Springer.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In *ICLR* (pp. 1–23). URL http://arxiv.org/abs/1511.05952, arXiv:1511.05952.

Schmidhuber, J. H. (1991). Curious model-building control systems. In *IEEE international joint conference on neural networks, vol. 2* (pp. 1458–1463). http://dx.doi.org/10.1109/IJCNN.1991.170605.

Schmidhuber, J. H. (1995). Environment-independent Reinforcement Acceleration.

Schmidhuber, J. H. (1999). A general method for incremental self-improvement and multi-agent learning. In X. Yao (Ed.), *Evolutionary computation: theory and applications, vol. 1* (pp. 81–123). World Scientific.

Schmidhuber, J. H. (2004). Optimal ordered problem solver. *Machine Learning*, *54*, 211–254.

Schmidhuber, J. H. (2007). Gödel Machines : fully self-referential optimal universal self-improvers. In B. Goertzel, & C. Pennachin (Eds.), *Artificial general intelligence* (pp. 199–226). Springer.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, *61*(January), 85–117. http://dx.doi.org/10.1016/j.neunet.2014.09.003, arXiv:1404.7828.

Schmidhuber, J. H., & Zhao, J. (1997). Multi-agent learning with the success-story algorithm. In *1221 LNAI, Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)* (pp. 82–93). http://dx.doi.org/10.1007/3-540-62934-3_43.

Schmidhuber, J. H., Zhao, J., & Schraudolph, N. N. (1997). Reinforcement learning with self-modifying policies. In S. Thrun, & L. Pratt (Eds.), *Learning to learn* (pp. 293–309). Kluwer Academic Publishers.

Schmidhuber, J. H., Zhao, J., & Wiering, M. (1996). Simple principles of metalearning. In *Technical report IDSIA-69-96* (pp. 1–23). Lugano: IDSIA.

Schmidhuber, J. H., Zhao, J., & Wiering, M. (1997). Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, *28*, 105–130. http://dx.doi.org/10.1023/A:1007383707642.

Shani, G., Pineau, J., & Kaplow, R. (2013). A survey of point-based pomdp solvers. *Autonomous Agents and Multi-Agent Systems*, *27*(1), 1–51. http://dx.doi.org/10.1007/s10458-012-9200-2.

Shapiro, S. C., & Rapaport, W. J. (1992). The SNePS family. *Computers Mathematicae Applicatae*, *23*(2), 243–275.

Sharma, S. K., & Chandra, P. (2010). Constructive neural networks: a review. *International Journal of Engineering Science and Technology*, *2*(12) 7847–7855.

Shibata, K. (2017). Functions that Emerge through End-to-End Reinforcement Learning - The Direction for Artificial General Intelligence - . URL http://arxiv.org/abs/1703.02239, arXiv:1703.02239.

Shibata, K., & Goto, K. (2013). Emergence of flexible prediction-based discrete decision making and continuous motion generation through actor-q-learning. In *2013 IEEE 3rd joint international conference on development and learning and epigenetic robotics, ICDL 2013 - electronic conference proceedings* (pp. 2–7). http://dx.doi.org/10.1109/DevLrn.2013.6652559.

Shibata, K., Nishino, T., & Okabe, Y. (2001). Actor-Q based active perception learning system. In *Proceedings - IEEE international conference on robotics and automation, vol. 1* (pp. 1000–1005). Seoul, Korea: http://dx.doi.org/10.1109/ROBOT.2001.932680.

Silver, D., & Veness, J. (2010). Monte-carlo planning in large POMDPs. *Advances in neural information processing systems (NIPS)*, 1–9, URL http://papers.nips.cc/paper/4031-monte-carlo-planning-in-large-pomdps/.

Silver, D. L., Yang, Q., & Li, L. (2013). Lifelong machine learning systems : beyond learning algorithms. In *AAAI spring symposium series* (pp. 49–55).

Singh, S. P., Barto, A. G., & Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, 1281–1288.

Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A., & Ignateva, A. (2015). Deep Attention Recurrent Q-Network, pp. 1–7, arXiv:1512.01693.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*(2), 99–127. http://dx.doi.org/10.1162/106365602320169811.

Storck, J., Hochreiter, S., & Schmidhuber, J. (1995). Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the international conference on artificial neural networks (ICANN95), vol. 2* (pp. 159–164).

Sun, R., & Zhang, X. (2004). Top-down versus bottom-up learning in cognitive skill acquisition. *Cognitive Systems Research*, *5*, 63–89. http://dx.doi.org/10.1016/j.cogsys.2003.07.001.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 3104–3112, URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural arXiv:1409.3215.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction*. MIT Press.

Thrun, S. B., & Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, *15*, 25–46.

University of Southampton (2017). The Iridis Compute Cluster, URL https://www.southampton.ac.uk/isolutions/staff/iridis.page.

Vamplew, P., & Ollington, R. (2005). On-line reinforcement learning using cascade constructive neural networks. In R. Khosla, R. J. Howlett, & L. C. Jain (Eds.), *9th international conference, KES 2005* (pp. 562–568). Springer.

Wang, P. (2007). The logic of intelligence. In *Artificial General Intelligence* (pp. 31–62). Springer.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292. http://dx.doi.org/10.1007/BF00992698, URL http://link.springer.com/10.1007/BF00992698.

Whitehead, S. D., & Ballard, D. H. (1990). Active perception and reinforcement learning. *Neural Computation*, *2*, 409–419.

Wierstra, D., Forster, A., Peters, J., & Schmidhuber, J. H. (2010). Recurrent policy gradients. *Logic Journal of IGPL*, *18*(5), 620–634.

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. http://dx.doi.org/10.1109/4235.585893.

Yang, S., Gao, Y., An, B., Wang, H., & Chen, X. (2016). Efficient average reward reinforcement learning using constant shifting values. In *Proceedings of the thirtieth AAAI conference on artificial intelligence (AAAI-16)* (pp. 2258–2264).

Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method, arXiv, p. 6, URL http://arxiv.org/abs/1212.5701, arXiv:1212.5701.

Zeng, S. Q., Tham, K. Y., Badgero, M., & Weng, J. Y. (2002). Dav : a humanoid robot platform for autonomous mental development. In *Proceedings of the 2nd IEEE conference on development and Learning* (pp. 73–81).

Zhao, J. (2002). Self-modifying reinforcement learning. In *Proceedings of the first international conference on machine learning and cybernetics* (pp. 1–38). Beijing, China.

Zhao, J., & Schmidhuber, J. H. (1998). Solving a complex prisoner ' s dilemma with self-modifying policies. In *From animals to animats 5: Proc. 5th international conference on simulation of adaptive behavior* (pp. 177–182). Zurich, Switzerland.

Appendix B

Experimental parameters

This appendix includes parameter settings and other implementation details required to reproduce the findings in the experiments.

B.1 Learning with limited knowledge and sparse rewards

This section describes the parameter settings used for the experiments in Chapter 4.

B.1.1 Learning parameters

For DRQN, all parameter settings, mentioned in Table B.1, were the same as in [69], except the *unroll* parameter, the trace-length for prediction and backpropagation through time, was set to 25 and 40 for easy and difficult mazes, respectively. The only exception is that the exploration frame of 10^6 time steps used in DRQN, in which the exploration rate is decreased linearly from $\epsilon = 1$ to $\epsilon = .10$, is not required for SMP-DRQN, since (a) the SMP does not necessarily rely on the Q-network; and (b) the exploration rate is controlled by the instruction module via arguments to `doQuntil`. The batch size and exploration rate were adapted dynamically by SMP-DRQN, starting initially from a uniform distribution with the same average as the original DRQN setting, namely, batchsize $= 32$ and $\epsilon = .10$.

For the SMPs using Incremental Self-improvement, the number of program cells m was set to 50 for easy and 100 for difficult problems. A minimal probability $minP = .0005$ ensured all instructions were regularly computed. The total number of working memory cells N_{wm}, including input, working, register and output cells, was 130 for easy and 220 for difficult problems. A small change was made to IS to encourage learning over the lifetime, namely, duplicates of each of the self-modification instructions `incP` and `decP` were added to the instruction set $\mathcal{A}$. For the SMP condition, both were duplicated 10

Table B.1: *DRQN parameters.*

Parameter	**Setting**
unroll	25 (easy), 40 (difficult)
batch size	32
replay memory size	400000 experiences
initial exploration rate	1.0
final exploration rate	0.1
exploration frame	1000000 time steps
optimisation algorithm	AdaDelta [238]
learning rate	0.1
momentum	0.95
clip gradient	absolute value exceeding 10
replay start	50000 time steps
update frequency	once every 4 time steps
target update frequency	once every 10000 updates

times, yielding $|\mathcal{A}| = 39$ and 22 modification instructions; For the SMP-Fixed condition, both were duplicated 9 times resulting in $|\mathcal{A}| = 39$ instructions, 21 of which were modification instructions; For the SMP-Constructive, both were duplicated 8 times resulting in $|\mathcal{A}| = 39$ instructions, 21 of which were modification instructions; For the SMP-DRQN, both were duplicated 9 times resulting in $|\mathcal{A}| = 40$, 22 of which were modification instructions. The effect of duplication is dicussed in Appendix C. In SMP-Constructive, addition of nodes was limited to a maximum of $max = 2MaxInt$ neurons, leading to 176 for easy and 276 for difficult problems. Using the above information, the following parameters were set to determine the addresses of the working memory: $Max = \mathcal{A} + m$; $Min = Max - N_{wm}$; $RegisterStart = 0$; $InputEnd = Min + N_{obs} + 4$, where $N_{obs} = 4$ cells record the observation. The input cells had addresses in $Min, \dots, InputEnd$, the working cells had addesses in $InputEnd + 1, \dots, RegisterStart - 1$, the register cells had addresses in $RegisterStart, \dots, |\mathcal{A}| - 1$, and the output cells had addresses in $|\mathcal{A}, Max|$. The range of representable numbers $[-MaxInt, MaxInt]$ was set using $MaxInt = \max(|Min|, Max)$ where Min is the lowest address and Max is the highest address in the working memory.

B.1.2 Maze generation procedure

The maze generation procedure is shown in Algorithm B.1.

Algorithm B.1 Procedure for generating mazes. Note: / is the integer division.

```
$sizeX \leftarrow 13$; $sizeY = 13$;
$compl \leftarrow .10$; $density \leftarrow .10$;
$length \leftarrow floor(complexity * (5 * (sizeX + sizeY)))$;
$islands \leftarrow floor(density * ((sizeX/2) * (sizeY/2)))$;
Fill borders with obstacles;
for $i \leftarrow 0$ to $islands - 1$ do
    $(y, x) \leftarrow$ get-random-position();
    set obstacle on (y,x);
    for $j \leftarrow 0$ to $length - 1$ do
        $neighbours \leftarrow \emptyset$;
        if $x > 1$ then
            $neighbours$.append($(y, x - 2)$;
        end if
        if $x < sizeX - 2$ then
            $neighbours$.append($(y, x + 2)$);
        end if
        if $y > 1$ then
            $neighbours$.append($(y - 2, x)$);
        end if
        if $y < sizeY - 2$ then
            $neighbours$.append($(y + 2, x)$);
        end if
        if length($neighbours$) $> 0$ then
            $\tilde{y}, \tilde{x} \leftarrow$ random-neighbour();
            if $(\tilde{y}, \tilde{x})$ is free then
                set obstacle on $(\tilde{y}, \tilde{x})$;
                $\bar{y} = \tilde{y} + (y - \tilde{y})/2$;
                $\bar{x} = \tilde{x} + (x - \tilde{x})/2)$;
                set obstacle on $(\bar{y}, \bar{x})$;
                $(y, x) \leftarrow (\tilde{y}, \tilde{x})$
            end if
        end if
    end for
end for
Pick start $S$ and goal $G$ manually.
if condition=$Random$ then
    Initialise $\alpha < 1, \beta > 1$
    $reachable \leftarrow$ reachable-from($G$);
    $d_{ref} \leftarrow$ dist(S,G);
    $starts \leftarrow \emptyset$
    for $p \in reachable$ do
        if dist($p, G$) $\in [\alpha d_{ref}, \beta d_{ref}]$ then
            $starts$.append($p$);
        end if
    end for
end if
```

B.1.3 Resulting mazes

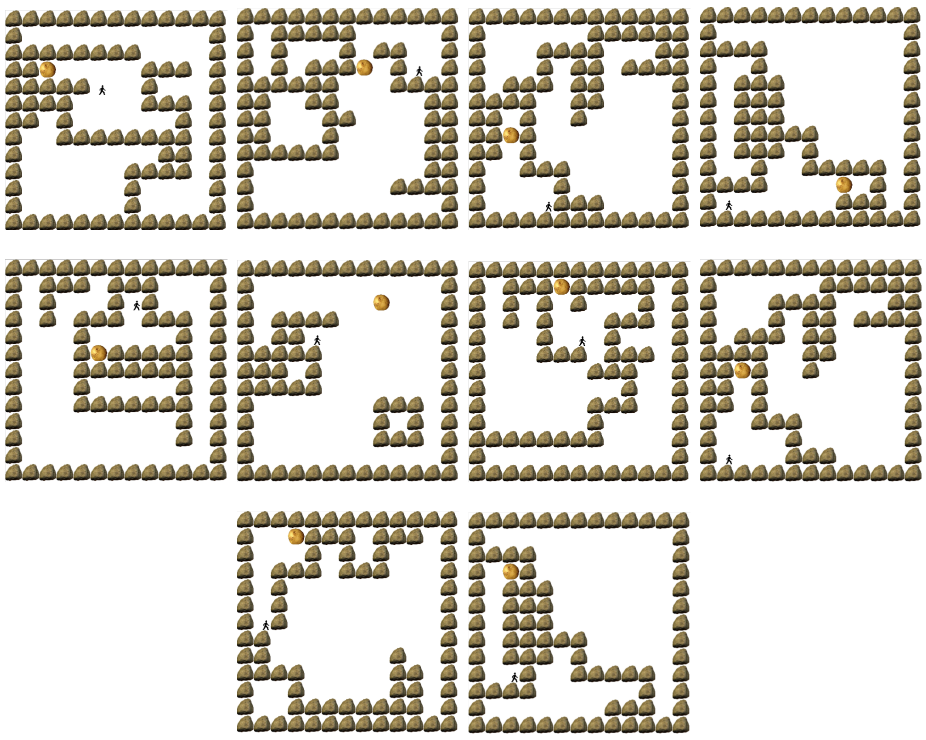


Figure B.1: *Complete set of easy mazes, which have an optimal solution of 4-9 steps. The yellow food source indicates the goal location while the agent is located on the starting position used for the easy-fixed condition.*

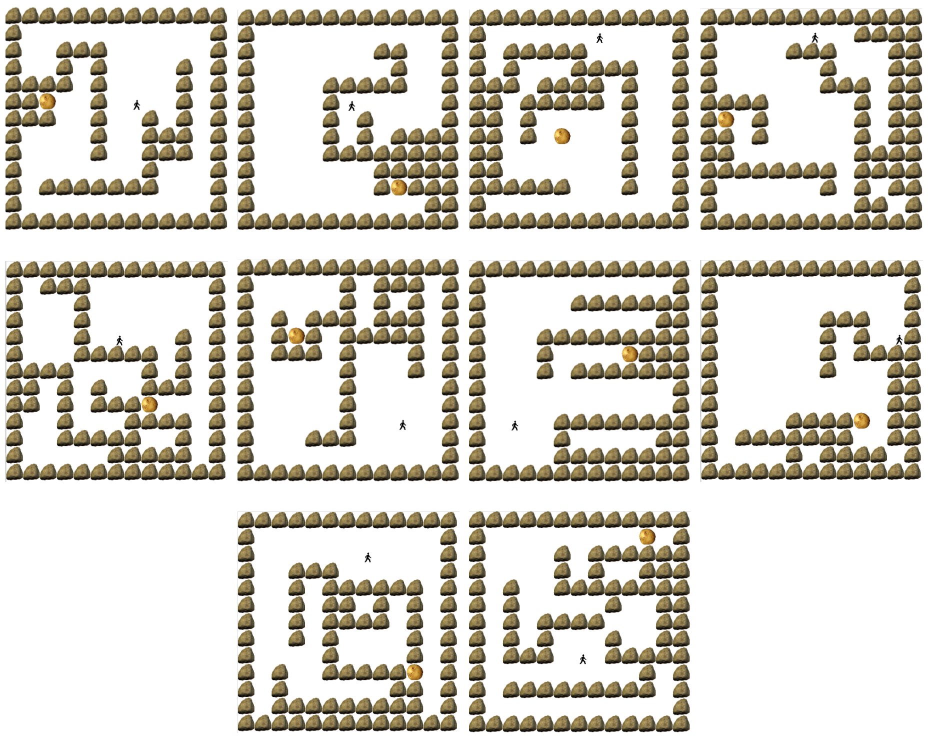


Figure B.2: *Complete set of difficult mazes, which have an optimal solution of 11-30 steps. The yellow food source indicates the goal location while the agent is located on the starting position used for the difficult-fixed condition.*

B.2 Lifelong learning with multiple policies

This section describes the parameter settings used for the experiments in Chapter 5.

B.2.1 Starting coordinates

Taking x increasing to the east and y increasing to the south, static main objects were initialised randomly from a set of coordinates: $\{(3,3),(5,3)\}$ in the cheese maze; $\{(6,1),(9,1),(9,6)\}$ in Sutton's maze; $\{(1,1),(1,7),(7,1),(7,7)\}$ in the pacman topology. This implies a general search strategy should be used, rather than the memorisation of a single path. Similar to the orginal tasks, the learner's initial position is $(1,2)$ for the cheese-maze, $(1,3)$ in Sutton's maze, and $(4,7)$ for the pacman task. The initial location of dynamic objects, similar to pacman ghosts, is based on a single home position, which is the central bottom location $(3,3)$ for the cheese maze, the top-right location $(9,1)$ for Sutton's maze, and the above-center location $(4,3)$ for the pacman topology.

B.2.2 Base learners

Code from `https://github.com/flyyufelix/VizDoom-Keras-RL/blob/master/` and `https://github.com/magnusja/ppo/` are taken as a template, and then modified to use with multiple policies, and to match the original papers [69, 169]: for DRQN, a slowly changing target value-function; for PPO, the set-up similar to the Atari experiments in [169], where actions are discrete, non-output layers are shared and the objective include additional terms for the value function and the entropy.

Parameter settings, mentioned in Table B.2 are chosen based on the original papers and any other modifications are chosen based on a limited tuning procedure, with the only exceptions being that (a) PPO used only a single actor to compare the various learners on the same number of experiences, to limit computational expense, and in line with real-life experiments where only one copy of the environment exists; (b) no annealing of parameters was done to keep the various conditions comparable. For both learners, the network topology is modified to suit the domain: an LSTM layer [77] was used for learning in partially observable environments, and convolutional layers are replaced by a single densely connected layer due to the small observation without spatial correlations.

B.3 Towards active adaptive perception in lifelong learning environments

This section describes the parameter settings used for the experiments in Chapter 6.

Table B.2: *Parameter settings for the base learners. "All" denotes settings common to all base-learners.*

Base Learner	**Parameter**	**Setting**
All	unroll	15
	hidden layers	80 (Dense RELU) - 80 (LSTM tanh)
	discount	0.99
DRQN	batch size	32
	update frequency	once every 4 time steps
	replay memory size	400000 experiences
	optimisation algorithm	AdaDelta [238]
	momentum	0.95
	learning rate	0.1
	exploration rate	0.2
	clip gradient	absolute value exceeding 10
	replay start	50000 time steps
	target update frequency	once every 10000 time steps (DRQN)
PPO	actors	1
	optimisation algorithm	Adam [94]
	learning rate	0.00025
	batch size	34
	update frequency	once every 100 time steps
	epochs	3
	clip gradient	norm exceeding 1.0
	GAE parameter	0.95
	clipped objective coefficient	0.10
	value function coefficient	1
	entropy coefficient	0.01

B.3.1 Learning parameters

The parameters for DRQN are similar to the above, with the following exceptions: the unroll parameter is lower to reduce computation time and because environments are not so large; the number of neurons in each hidden layer was set to 80 instead of 50, because larger networks can potentially represent more tasks. AdaDelta is comparatively robust to various hyperparameter settings so the learning rate is taken from earlier studies [24, 69]; the final exploration rate was set higher, to $\epsilon = .20$, than the $\epsilon = .10$ in [24, 69], and compared to these works the final exploration rate was used throughout the lifetime without any annealing schedule. This choice (a) allows a greater flexibility throughout the lifetime; (b) avoids overexploiting faulty policies when a new task arrives. Table B.4 is not the basis of the parameter choice, but illustrates how, on the first three scenarios, $\epsilon = .10$ and $\epsilon = .20$ give the strongest results in the first two million time steps of the lifetime. To exploit the temporal structure, DRQN's buffer is organised in episodes. The same topology is used as in the previous study.

For the SSA agents, the initial uniform distribution of the hyperparameters is such that the baseline DRQN's setting was the mean: $[0, 0.40]$ for the exploration rate; $[10^{-6}, 0.20]$ for the learning rate. The SSA agents are then allowed to modify the distribution freely, although the bounds remain the same across the lifetime. Working memory variables such as Max, $MaxInt$, $InputEnd$, etc. are obtained in the same manner as described in Section B.1.1, although some of the settings are different: the working memory cells $N_{wm} = 180$ and $N_{obs} = 11$. The instruction set $\mathcal{A}$ consisted of 28 instructions, and

Table B.3: *DRQN parameters.*

Parameter	**Setting**
unroll	15
batch size	32
replay memory size	400000 experiences
exploration rate	0.2
learning rate	0.1
optimisation algorithm	AdaDelta [238]
momentum	0.95
clip gradient	absolute value exceeding 10
replay start	50000 time steps
update frequency	once every 4 time steps
target update frequency	once every 10000 updates

Table B.4: *Effect of exploration rate on performance.*

Parameter setting	**Cumulative reward**
$\epsilon = .05$	34689 ± 133302
$\epsilon = .10$	99351 ± 178464
$\epsilon = .20$	91926 ± 166078

the self-modification instructions were duplicated 5 times each, resulting in 12 self-modification instructions and 2 instructions related to network changes.

Appendix C

Additional results (Chap. 4)

C.1 Effect of duplication

Duplication of `incP` and `decP` is suggested to improve the performance of Incremental Self-improvement. Comparative results with and without duplication are shown in Table C.1. They illustrate that, even without duplication, the active adaptive perception methods always outperforms the traditional SMP in the difficult environments. However, duplication provides an additional positive effect on performance. This positive effect is attributed to a greater flexibility in change sizes. For example, if a particular entry has .001 as a probability, performing `decP` on this entry shrinks the entry to between .00001 and .00099, whereas performing `decP` on another entry with a probability .10 will decrease its probability to between .001 and .099, which is a much larger absolute decrease. Moreover, there are favourable side-effects: (a) there is a higher initial probability of self-modification instructions since they have multiple entries in the matrix; (b) the combined minimal probability of self-modification is higher since each entry in the probability matrix must have a probability greater than $minP$; (c) an enhanced syntactical correctness, resulting in more correct executions of the self-modification instructions.

Table C.1: *Effect of duplication of the `incP` and `decP` instructions on the lifetime average of the normalised reward speed. Bold font is used to illustrate the best-performing learner without duplication. A * sign indicates the best-perfoming learner in general, duplication or no duplication.*

Condition	**Learner**	**Dupl**	**No Dupl**
Easy-Fixed	SMP	.313	**.362***
	SMP-Fixed	.325	.323
	SMP-Constructive	.312	.306
Easy-Random	SMP	.284	.257
	SMP-Fixed	.308	.258
	SMP-Constructive	.314*	**.272**
Difficult-Fixed	SMP	.023	.023
	SMP-Fixed	.069*	.041
	SMP-Constructive	.056	**.048**
Difficult-Random	SMP	.021	.018
	SMP-Fixed	.054*	.039
	SMP-Constructive	.050	**.051**

Appendix D

Additional results (Chap. 5)

D.1 Ordinal analysis

The following two pages include the ordinal analyses on the impact of adaptivity and the number of policies.

Table D.1: *Ordinal analysis on the effect of adaptivity and the number of policies on performance (median ± interquartile range) of DRQN. Performance is based on the lifetime reward velocity $R(T)/T$ where $T = 9 * 10^7$ is the total lifetime of the learner.* ***significance*** *denotes the p-value obtained from the Wilcoxon ranksum-test.* ***effect size*** *denotes Cliff's δ, an ordinal effect size metric for dominance of one distribution over the other. The label in parenthesis denotes the magnitude of the effect; its estimate is based on Vargha et al.'s study [221].*

Method	**Performance**	**Comparison**													
		AdaptiveDRQN4P		AdaptiveDRQN9P		UnadaptiveDRQN1P		UnadaptiveDRQN2P		UnadaptiveDRQN4P		UnadaptiveDRQN9P		UnadaptiveDRQN18P	
		significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size
AdaptiveDRQN2P	0.099 ± 0.016	$p < 0.001^{**}$	-0.94 (large)	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	1.00 (large)	$p < 0.001^{**}$	0.95 (large)	$p = 0.527$	-0.12 (small)	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	-1.00 (large)
AdaptiveDRQN4P	0.156 ± 0.037	/	/	$p < 0.001^{**}$	-0.78 (large)	$p < 0.001^{**}$	1.00 (large)	$p < 0.001^{**}$	1.00 (large)	$p < 0.001^{**}$	0.76 (large)	$p < 0.001^{**}$	-0.90 (large)	$p < 0.001^{**}$	-1.00 (large)
AdaptiveDRQN9P	0.206 ± 0.038	/	/	/	/	$p < 0.001^{**}$	1.00 (large)	$p < 0.001^{**}$	1.00 (large)	$p < 0.001^{**}$	0.99 (large)	$p = 0.066$	-0.36 (medium)	$p < 0.001^{**}$	-1.00 (large)
UnadaptiveDRQN1P	0.015 ± 0.007	/	/	/	/	/	/	$p < 0.001^{**}$	-0.97 (large)	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	-1.00 (large)
UnadaptiveDRQN2P	0.047 ± 0.017	/	/	/	/	/	/	/	/	$p < 0.001^{**}$	-0.91 (large)	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	-1.00 (large)
UnadaptiveDRQN4P	0.102 ± 0.046	/	/	/	/	/	/	/	/	/	/	$p < 0.001^{**}$	-1.00 (large)	$p < 0.001^{**}$	-1.00 (large)
UnadaptiveDRQN9P	0.223 ± 0.042	/	/	/	/	/	/	/	/	/	/	/	/	$p < 0.001^{**}$	-0.99 (large)
UnadaptiveDRQN18P	0.310 ± 0.023	/	/	/	/	/	/	/	/	/	/	/	/	/	/

Table D.2: *Ordinal analysis on the effect of adaptivity and the number of policies on performance (median $\pm$ interquartile range) of PPO. Performance is based on the lifetime reward velocity $R(T)/T$ where $T = 9 * 10^7$ is the total lifetime of the learner.* ***significance*** *denotes the p-value obtained from the Wilcoxon ranksum-test.* ***effect size*** *denotes Cliff's δ, an ordinal effect size metric for dominance of one distribution over the other. The label in parenthesis denotes the magnitude of the effect; its estimate is based on Vargha et al.'s study [221].*

Method	**Performance**	**Comparison**													
		AdaptivePPO4P		AdaptivePPO9P		UnadaptivePPO1P		UnadaptivePPO2P		UnadaptivePPO4P		UnadaptivePPO9P		UnadaptivePPO18P	
		significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size	significance	effect size
AdaptivePPO2P	0.060 ± 0.013	$p = 0.591$	-0.10 (negligible)	$p = 0.146$	-0.28 (medium)	$p = 0.229$	0.23 (small)	$p = 0.704$	0.07 (negligible)	$p = 0.002^{*}$	-0.59 (large)	$p < 0.001^{**}$	-0.74 (large)	$p < 0.001^{**}$	-0.86 (large)
AdaptivePPO4P	0.065 ± 0.010	/	/	$p = 0.376$	-0.17 (small)	$p = 0.054$	0.38 (medium)	$p = 0.359$	0.18 (small)	$p = 0.008^{*}$	-0.52 (large)	$p < 0.001^{**}$	-0.67 (large)	$p < 0.001^{**}$	-0.84 (large)
AdaptivePPO9P	0.067 ± 0.031	/	/	/	/	$p = 0.016^{*}$	0.47 (large)	$p = 0.129$	0.30 (medium)	$p = 0.082$	-0.34 (medium)	$p = 0.046^{*}$	-0.39 (medium)	$p = 0.011^{*}$	-0.49 (large)
UnadaptivePPO1P	0.057 ± 0.010	/	/	/	/	/	/	$p = 0.311$	-0.20 (small)	$p < 0.001^{**}$	-0.74 (large)	$p < 0.001^{**}$	-0.85 (large)	$p < 0.001^{**}$	-0.93 (large)
UnadaptivePPO2P	0.059 ± 0.013	/	/	/	/	/	/	/	/	$p = 0.002^{*}$	-0.59 (large)	$p < 0.001^{**}$	-0.69 (large)	$p < 0.001^{**}$	-0.80 (large)
UnadaptivePPO4P	0.081 ± 0.028	/	/	/	/	/	/	/	/	/	/	$p = 0.411$	-0.16 (small)	$p = 0.217$	-0.24 (small)
UnadaptivePPO9P	0.084 ± 0.027	/	/	/	/	/	/	/	/	/	/	/	/	$p = 0.776$	-0.06 (negligible)
UnadaptivePPO18P	0.086 ± 0.022	/	/	/	/	/	/	/	/	/	/	/	/	/	/

Appendix E

Additional results (Chap. 6)

E.1 Lifelong SSC proof

Theorem E.1. *Algorithm 6.2 maximises the global velocity, such that the Lifelong Success Story Criterion in Equation 6.1 is satisfied.*

Proof via induction: It will be shown that (a) initially after the first evaluation $Firsts.top() = \arg\max_{e \in Firsts \subset \mathscr{S}} \tilde{V}(e)$; and then (b) if the statement is true for evaluation i, it is also true for evaluation $i+1$. For notation, the abbreviation $top = Firsts.top()$ and $second = Firsts.secondtop()$ will be used for the top and second to top entry in $Firsts$.

(a) At the first evaluation, the only entries in $Firsts$ are the initial stack entry e_0, which has task time and reward equal to zero for all tasks, and top, and $second = e_0$. With only the current task $\mathbf{F}$ seen so far, $V_{\mathbf{F}} = \tilde{V}$. Therefore, if $\tilde{V}(second) \geq \tilde{V}(top)$, the algorithm will set $maxfirst$ to the $second$ entry, and then pops back such that e_0 is now the only entry and $top = e_0$; it then follows $top = \arg\max_{e \in Firsts \subset \mathscr{S}} \tilde{V}(e)$. If $\tilde{V}(second) < \tilde{V}(top)$ then the algorithm will return immediately, resulting in $top = \arg\max_{e \in Firsts \subset \mathscr{S}} \tilde{V}(e)$.

(b) Suppose the previous evaluation point satisfied $top = \arg\max_{e \in Firsts \subset \mathscr{S}} \tilde{V}(e) = \sum_j C_{j=1}^{N}(e)$; after the new self-modification sequence this entry can be fetched as $second$. The new self-modification sequence may either improve on the global maximum or not, and there are four special cases that deserve special attention:
(i) If the new self-modification sequence top, satisfies $V_{\mathbf{F}}(top) > V_{\mathbf{F}}(second)$, then since only the current task $\mathbf{F}$ has been affected in this evaluation interval and previously $\tilde{V}(second)$ was maximal, this means $\tilde{V}(top)$ is now maximal.
(ii) Suppose that the new self-modification sequence did not improve on the previous.
(ii).1: entries with $\Delta\tilde{V}(e) < 0 \land -\Delta\tilde{V}(second) < \epsilon$. The maximum cannot have changed for sufficiently small $\epsilon > 0$, because previously $second$ was the maximum.

(ii).2: entries with $\Delta\tilde{V}(e) < 0 \wedge -\Delta\tilde{V}(second) \geq \epsilon$. The previous baseline-adjusted global velocity's maximum was entry *second*; since $-\Delta\tilde{V}(second) \geq \epsilon$, earlier entries could now have $\tilde{V}(e) \geq \tilde{V}(second)$. Therefore, checking all entries $e \in Firsts$ which satisfy $-\Delta\tilde{V}(e) \geq \epsilon$ is sufficient to find the baseline-adjusted global velocity's maximum Since $\Delta V_{\mathbf{F}}(e) = V_{\mathbf{F}}(e) - V_{\mathbf{F}}^{old}(e)$, an upper bound on $\Delta V_{\mathbf{F}}(e)$ can be obtained as follows:

$$\begin{aligned}
-\Delta V_{\mathbf{F}}(e) &= V_{\mathbf{F}}^{old}(e) - \frac{top.t_{\mathbf{F}} - e.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}} V_{\mathbf{F}}^{old}(e) - \frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}} V_{\mathbf{F}}(top) && \text{(E.1)}\\
&= \frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}} (V_{\mathbf{F}}^{old}(e) - V_{\mathbf{F}}(top)) && \text{(E.2)}\\
&\leq \frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}} (\max(V_{\mathbf{F}}^{old}) - V_{\mathbf{F}}(top)) && \text{(E.3)}\\
&. && \text{(E.4)}
\end{aligned}$$

Note that, as $t_{\mathbf{F}} - e.t_{\mathbf{F}}$ grows larger going further down the stack, the upper bound on $-\Delta V_{\mathbf{F}}(e)$ shrinks, with $t_{\mathbf{F}} - e.t_{\mathbf{F}} \rightarrow \infty$ implying $-\Delta V_{\mathbf{F}}(e) \rightarrow 0$. This progressive shrinking implies that searching entries $e \in Firsts$ until :

$$\begin{aligned}
-\Delta\tilde{V}(e) &= -\frac{w}{S}\Delta V_{\mathbf{F}}(e) && \text{(E.5)}\\
&\leq \frac{w}{S}\frac{t_{\mathbf{F}} - top.t_{\mathbf{F}}}{t_{\mathbf{F}} - e.t_{\mathbf{F}}} (\max(V_{\mathbf{F}}^{old}) - V_{\mathbf{F}}(top)) < \epsilon \quad , && \text{(E.6)}
\end{aligned}$$

is sufficient to find the maximum, given a user-defined tolerance of error $\epsilon > 0$. Note that to assess whether or not case (ii).2 holds, it is required to know whether $\Delta\tilde{V}(e) < 0$. For this purpose, it is sufficient to check that $V_{\mathbf{F}}(e) > V_{\mathbf{F}}(top)$ [1]. The procedure assumes an upper bound for the task velocity can be found which satisfies $\hat{M}_{\mathbf{F}} \geq \max(V_{\mathbf{F}}^{old})$. The estimated maximum task velocity is reset regularly to the actual maximum, $\hat{M} = \max(V_{\mathbf{F}})$, after the popping procedure in cases (iii) and (iv) [2] From there on, it is (over)estimated at the start of each cycle: if $V_{\mathbf{F}}(top) \leq \hat{M}_{\mathbf{F}}$, $\hat{M}_{\mathbf{F}} \leftarrow \frac{t_M}{t_M + \Delta t}\hat{M}_{\mathbf{F}} + \frac{t_M}{\Delta t}V_{\mathbf{F}}(top)$ and $t_M \leftarrow t_M + \Delta_t$; if $V_{\mathbf{F}} > \hat{M}_{\mathbf{F}}$, $\hat{M}_{\mathbf{F}} \leftarrow V_{\mathbf{F}}(top)$ and $t_M \leftarrow \Delta t$.

(ii).3: entries with $\Delta\tilde{V}(e) \geq 0$.. If an entry satisfies $\Delta\tilde{V}(e) \geq 0$, then this means that $\tilde{V}^{old}(e) \leq \tilde{V}(top)$. Because $\tilde{V}(e)$ will be an interpolation between $\tilde{V}^{old}(e)$ and $\tilde{V}(top)$, this means that $\tilde{V}(e) < \tilde{V}(top) \leq \tilde{V}(second)$.

(iii) When a parameter changed, this affects all entries without any progressively declining effect. Therefore, if during the new evaluation interval there is also a parameter change, the algorithm simply evaluates all entries and this way the maximum is always found.

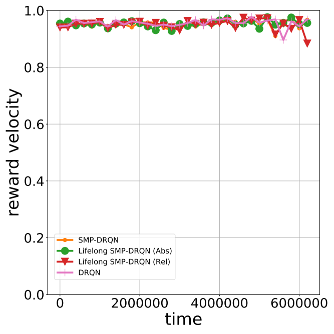
(iv) When the current task was previously unseen by the algorithm, its baseline was $B^{old} = 0$, and its component is $C^{old} = \frac{w^{old}}{S^{old}}(V_{\mathbf{F}}^{old} - B^{old}) = 0$, with $B^{old} = 0$. Therefore, any entry in the stack will have $V_{\mathbf{F}}^{old} = 0$, and the first self-modification sequence will

[1]Since $V_{\mathbf{F}}(e)$ is an interpolation between $V_{\mathbf{F}}^{old}(e)$ and $V_{\mathbf{F}}(top)$, this implies $V_{\mathbf{F}}(e) > V_{\mathbf{F}}(top) \iff V_{\mathbf{F}}^{old}(e) > V_{\mathbf{F}}(e)$.

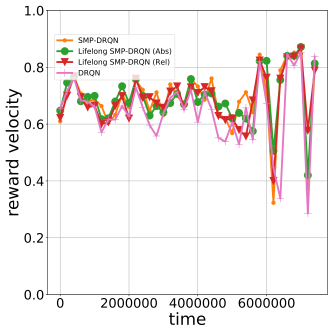
[2]this adds no computation because in these cases $\tilde{V}$ needs to be computed for all entries, and this requires $V_{\mathbf{F}}$.

change $\tilde{V}(e)$ of earlier entries according to $\Delta\tilde{V}(e) = V_{\mathbf{F}}(e)$. As in (iii), this case implies there is no progressively declining effect as one traverses the stack from top to bottom, and is resolved by evaluating all entries.

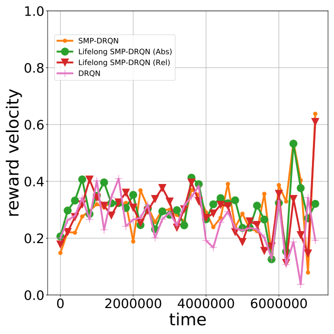
E.2 Task-specific development plots



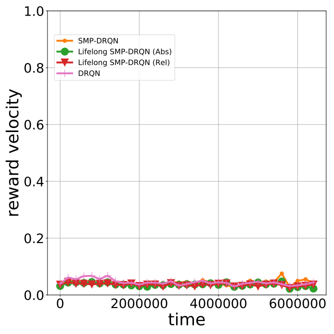
(a) $\mathbf{F} = (-1, 0, 2)$



(b) $\mathbf{F} = (-1, 2, 0)$



(c) $\mathbf{F} = (1, 0, 0)$



(d) $\mathbf{F} = (1, 2, 2)$

Figure E.1: *Performance development of the various evaluation modules on individual tasks. Since the lifelong scenario involves different blocks of tasks, the different blocks of each unique task are here glued together, illustrating the total time spent in the task on* x*-axis, and the reward velocity on that task on* y*-axis. The task is indicated in the subcaption as* $\mathbf{F}$*, with values along three dimensions: the reward incurred by the object* $\{-1, 1\}$*, the dynamicity of the object* $\{0, 1, 2\}$*, and the topology of the environment* $\{0, 1, 2\}$*. Performance is the reward velocity normalised such that 0 indicates obtaining the minimal reward at every time step and 1 indicates obtaining the maximal reward at every time step; these are under- and over-estimates of the worst and best performance, respectively.*

References

[1] Ackerman, E. (2018). OpenAI Releases Algorithm That Helps Robots Learn from Hindsight. Retrieved on 27 October 2019 from https://spectrum.ieee.org/automaton/robotics/artificial-intelligence/openai-releases-algorithm-that-helps-robots-learn-from-hindsight.

[2] Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. E. (2014). Online multi-task learning for policy gradient methods. *31st International Conference on Machine Learning, ICML 2014*, 4:2949–2957.

[3] Anderson, J. R. (1976). *Language, memory, and thought.* Erlbaum, Hillsdale, N.J.

[4] Anderson, J. R. (1983). *The architecture of cognition.* Harvard University Press., Cambridge, Massachusetts.

[5] Andrychowicz, M., Denil, M., Colmenarejo, S. G., and Hoffman, M. W. (2016). Learning to learn by gradient descent by gradient descent. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, pages 1–17.

[6] Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Processing Magazine, Special Issue on Deep Learning for image understanding*, 34:26–38.

[7] Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., and Yoshida, C. (2009). Cognitive Developmental Robotics: A Survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34.

[8] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the Multiarmed Bandit Problem. *Machine Learning*, (47):235–256.

[9] Baird, L. (1999). *Reinforcement learning through gradient descent.* Phd thesis, Carnegie Mellon University.

[10] Baird, L. C. (1994). Reinforcement learning in continuous time: advantage updating. In *Proceedings of the International Conference of Neural Networks*, Orlando, Florida.

[11] Bajcsy, R., Aloimonos, Y., and Tsotsos, J. K. (2017). Revisiting active perception. *Autonomous Robots*, 42(2):177–196.

[12] Bakker, B. (2002). Reinforcement Learning with Long Short-Term Memory. In *Advances in Neural Information Processing Systems 12 (NIPS 2002).*

[13] Baldi, P. and Vershynin, R. (2019). The capacity of feedforward neural networks. *Neural Networks*, 116:288–311.

[14] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-13(5):834–846.

[15] Baxter, J. (2000). A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research*, 12:149.

[16] Bellemare, M. G. and Dabney, W. (2017). A Distributional Perspective on Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, Sydney, Australia. PMLR.

[17] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain.

[18] Bellman, R. (1957a). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(4):679–684.

[19] Bellman, R. (1957b). *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA.

[20] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum Learning. In *Proceedings of the 26th International Conference on Machine Learning,*, Montreal, Canada.

[21] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

[22] Besold, T. R., d'Avila Garcez, A., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kuehnberger, K.-U., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. (2017). Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. In *Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches: Papers from the 2015 AAAI Spring Symposium*, pages 1–58. AAAI.

[23] Bieger, J., Thorisson, K. R., Steunebrink, B. R., Thorarensen, T., and Sigurdardottir, J. S. (2016). Evaluation of General-Purpose Artificial Intelligence : Why,

What & How. In *EGPAI 2016 - Evaluating General-Purpose A.I., Workshop held in conjuction with the European Conference on Artificial Intelligence*, The Hague, The Netherlands.

[24] Bossens, D. M., Townsend, N. C., and Sobey, A. J. (2019). Learning to learn with active adaptive perception. *Neural Networks*, 115:30–49.

[25] Bou Ammar, H., Eaton, E., Luna, J. M., and Ruvolo, P. (2015). Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. *IJCAI International Joint Conference on Artificial Intelligence*, 2015-Janua(Ijcai):3345–3351.

[26] Brunskill, E. and Li, L. (2014). PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 316–324, Beijing, China. JMLR: W&CP.

[27] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018a). Exploration by Random Network Distillation. *arXiv preprint*, pages 1–17.

[28] Burda, Y., Storkey, A., Darrell, T., and Efros, A. A. (2018b). Large-Scale Study of Curiosity-Driven Learning. *arXiv preprint*.

[29] Cair, O. (2011). External measures of cognition. *Frontiers in Human Neuroscience*, 5:1–9.

[30] Caruana, R. (1997). Multitask Learning. *Machine Learning*, 75(1):41–75.

[31] Cassimatis, N. L. (2002). *Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes.* Phd thesis, Massachusetts Institute of Technology.

[32] Chaplot, D. S. and Lample, G. (2017). Arnold: An Autonomous Agent to play FPS Games. *Thirty-First AAAI Conference on Artificial Intelligence. Foerster,*, pages 2–3.

[33] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation. In *Proceedings ofthe 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

[34] Choi, D. and Langley, P. (2018). Evolution of the ICARUS Cognitive Architecture. *Cognitive Systems Research*, 48:25–38.

[35] Chrabaszcz, P., Loshchilov, I., and Hutter, F. (2018). Back to basics: Benchmarking canonical evolution strategies for playing atari. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1419–1426.

[36] Cohen, N., Sharir, O., Tamari, R., and Shashua, A. (2017). Analysis and Design of Convolutional Networks. Technical Report May, Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI).

[37] Coop, R., Mishtal, A., and Arel, I. (2013). Ensemble learning in fixed expansion layer networks for mitigating catastrophic forgetting. *IEEE Transactions on Neural Networks and Learning Systems*, 24(10):1623–1634.

[38] Crawford, I. A. (2016). The long-term scientific benefits of a space economy. *Space Policy*, 37:58–61.

[39] Cully, A., Clune, J., Tarapore, D., and Mouret, J. B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503–507.

[40] Dai, A. and Le, Q. V. (2017). HyperNetworks. In *ICLR 2017*.

[41] d'Avila Garcez, A., Dutra, A. R. R., and Alonso, E. (2018). Towards Symbolic Reinforcement Learning with Common Sense. *arXiv preprint*, pages 1–15.

[42] Deisenroth, M. P., Englert, P., Peters, J., Fox, D., and Feb, M. L. (2014). Multi-Task Policy Search. In *IEEE International Conference on Robotics & Automation (ICRA)*, pages 3876–3881, Hong Kong, China.

[43] Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., and Maltoni, D. (2018). Don't forget, there is more than forgetting: new metrics for Continual Learning. In *Continual Learning Workshop at NeurIPS 2018*, pages 1–7, Montreal, Canada.

[44] Ecoffet, A., Stanley, K. O., and Clune, J. (2019). Go-Explore : a New Approach for Hard-Exploration Problems. *arXiv*, pages 1–37.

[45] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

[46] Fahlman, S. E. (1991). The recurrent cascade-correlation architecture. Technical report, School of Computer Science, Carnegie Mellon University.

[47] Farias, V. F., Moallemi, C. C., Roy, B. V., and Member, S. (2010). Universal Reinforcement Learning. *IEEE Transactions on Information Theory*, 56(5):2441–2454.

[48] Felix A. Gers, Schmidhuber, J. H., and Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

[49] Fernández, F. and Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS'06)*, page 720.

[50] Feurer, M. and Hutter, F. (2019). Hyperparameter Optimization. In Feurer, M. and Hutter, F., editors, *Automated Machine Learning*, chapter 1, pages 3–33.

[51] Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia.

[52] Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62.

[53] Folli, V., Leonetti, M., and Ruocco, G. (2017). On the Maximum Storage Capacity of the Hopfield Model. *Frontiers in Computational Neuroscience*, 10:1–9.

[54] Foster, M. (2018). Aging Japan: Robots may have role in future of elder care. Retrieved at 27 October 2019 from https://www.reuters.com/article/us-japan-ageing-robots-widerimage/aging-japan-robots-may-have-role-in-future-of-elder-care-idUSKBN1H33AB.

[55] Franco, L. and Conde, I. M. (2008). Active Learning Using a Constructive Neural Network Algorithm. In Kurkova, V., Neruda, R., and Koutnik, J., editors, *ICANN 2008, Part II, Lecture Notes in Computer Science 5164*, pages 803–811, Prague, Czech Republic. Springer-Verlag Berlin Heidelberg.

[56] Franklin, S., Madl, T., Mello, S. D., and Snaider, J. (2014). LIDA : A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development*, 6(1):19–41.

[57] Franklin, S. and Patterson, F. (2006). The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *Integrated Design and Process Technology*, pages 1–8.

[58] Frean, M. (1990). The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, 2(2):198–209.

[59] Friedland, G. and Krell, M. (2017). A Capacity Scaling Law for Artificial Neural Networks. *arXiv preprint*, pages 1–13.

[60] Garnelo, M., Arulkumaran, K., and Shanahan, M. (2016). Towards Deep Symbolic Reinforcement Learning. *arXiv preprint*, pages 1–13.

[61] Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38(1):173–198.

[62] Goertzel, B. (2010). Toward a Formal Characterization of Real-World General Intelligence. *Artificial General Intelligence: Proceedings of the Third Conference on Artificial General Intelligence, AGI 2010, Lugano, Switzerland, March 5-8, 2010*, pages 19–24.

[63] Goertzel, B. (2014). Artificial General Intelligence : Concept , State of the Art , and Future Prospects. *Journal of Artificial General Intelligence*, 5(1):1–48.

[64] Goertzel, B., Ke, S., Lian, R., Neill, J. O., Sadeghi, K., Wang, D., Watkins, O., and Yu, G. (2013). The CogPrime Architecture for Embodied Artificial General Intelligence. In *2013 IEEE Symposium on Computational Intelligence for Human-like Intelligence (CIHLI)*, pages 60–67.

[65] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 28:1319–1327.

[66] Graham, B. (2014). Spatially-sparse convolutional neural networks. *ArXiv [cs.CV]*, pages 1–13.

[67] Greff, K., Srivastava, R. K., Koutn, J., and Steunebrink, B. R. (2017). LSTM : A Search Space Odyssey. *Transactions on Neural Networks and Learning Systems*, pages 1–12.

[68] Harnad, S. (2003). The Symbol Grounding Problem. *Encyclopedia of Cognitive Science*, 42(1-3):335–346.

[69] Hausknecht, M. and Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium Series*, pages 29–37. AAAI.

[70] Haussler, D. (1993). The Probably Approximately Correct (PAC) and Other Learning Models. In Meyrowitz, A. L. and Chipman, S., editors, *Foundations of Knowledge Acquisition*, pages 291–312. Springer.

[71] Hawkins, J. (2004). *On intelligence.* Times Books.

[72] Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. (2015). Memory-based control with recurrent neural networks. *arXiv preprint*, pages 1–11.

[73] Heess, N., TB, D., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S. M. A., Riedmiller, M., and Silver, D. (2017). Emergence of Locomotion Behaviours in Rich Environments. *arXiv preprint.*

[74] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep Reinforcement Learning that Matters. *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3207–3214.

[75] Hessel, M., Modayil, J., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., and Silver, D. (2013). Rainbow : Combining Improvements in Deep Reinforcement Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3215–3222. AAAI.

[76] Hester, T. and Stone, P. (2013). TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3):385–429.

[77] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1–32.

[78] Hochreiter, S., Younger, A., and Conwell, P. (2001). Learning to learn using gradient descent. In *Lecture Notes in Computer Science 2130, Proc. Intl. Conf. on Artificial Neural Networks (ICANN-2001*, pages 87–94. Springer.

[79] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

[80] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. (2017). Snapshot Ensembles: Train 1, get M for free. In *Proceedings of the International Conference on Learning Representations (ICLR 2017)*, pages 1–14.

[81] Hutter, M. (2007). Universal Algorithmic Intelligence: A mathematical top-down approach. In B., G. and C., P., editors, *Artificial General Intelligence.* Springer, Berlin, Heidelberg.

[82] Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*, volume 4, pages 2588–2595, Canberra, ACT, Australia. IEEE.

[83] Ingle, S. (2018). Creative' AlphaZero leads way for chess computers and, maybe, science. Retrieved at 27 October from https://www.theguardian.com/sport/2018/dec/11/creative-alphazero-leads-way-chess-computers-science.

[84] Isele, D. and Cosgun, A. (2018). Selective Experience Replay for Lifelong Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3302–3309.

[85] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *ACM International Conference on Multimedia*, pages 675–678.

[86] Junior, B., Nicoletti, C., and Lu, R. W. (2016). Enhancing Constructive Neural Network performance using functionally expanded input data. *Journal of Artificial Intelligence and Soft Computing Research*, 6(2):119–131.

[87] Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134.

[88] Kakade, S. (2002). A Natural Policy Gradient. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063.

[89] Kamal, M. A. S., Murata, J., and Hirasawa, K. (2004). Task-oriented reinforcement learning for continuing task in dynamic environment. *Research Reports on Information Science and Electrical Engineering of Kyushu University*, 9(1):7–12.

[90] Karnowski, T. P., Arel, I., and Rose, D. (2010). Deep Spatiotemporal Feature Learning with Application to Image Classification. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 883–888.

[91] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Li, F. F. (2014). Large-scale video classification with convolutional neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1725–1732.

[92] Kemker, R., Mcclure, M., Abitino, A., Hayes, T. L., and Kanan, C. (2017). Measuring Catastrophic Forgetting in Neural Networks. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 3390–3398.

[93] Kieras, D. E. and Meyer, D. E. (1997). An Overview of the EPIC Architecture for Cognition and Performance With Application to Human-Computer Interaction. *Human-Computer Interaction*, 12:391–438.

[94] Kingma, D. P. and Ba, J. L. (2015). Adam: A method for Stochastic Optimisation. In *International Conference on Learning Representations (ICLR 2015)*, pages 1–15.

[95] Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., and Rusu, A. A. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114(13):3521–3526.

[96] Konidaris, G., Scheidwasser, I., and Barto, A. G. (2012). Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(1):1333–1371.

[97] Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. Technical report, Science Department, University of Toronto.

[98] Krizhevsky, A. and Hinton, G. E. (2010). Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, pages 1–9.

[99] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9.

[100] Laird, J. E. (2008). Extending the Soar cognitive architecture. *Proceedings of the 2008 conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, 171:224–235.

[101] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for General Intelligence. Technical report, University of Michigan, Carnegie-Mellon University, Stanford University.

[102] Lample, G. and Chaplot, D. S. (2017). Playing FPS Games with Deep Reinforcement Learning. In *Thirty-First AAAI Conference on Artificial Intelligence.*, pages 2140–2146.

[103] Langley, P. and Choi, D. (2006). A Cognitive Architecture for physical agents. *Proceedings Of The National Conference On Artificial Intelligence*, 21(2):1469.

[104] Lecun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521.

[105] Legg, S. and Hutter, M. (2007). Universal Intelligence: A Definition of Machine Intelligence. *Minds & Machines*, 17(4):391–444.

[106] Leike, J. and Hutter, M. (2015). Bad Universal Priors and Notions of Optimality. In *JMLR: Workshop and Conference Proceedings*, volume 40, pages 1–16.

[107] Lenat, D. B., Guha, R. V., Pittman, K., Pratt, D., and Shepherd, M. (1990). Cyc: toward programs with common sense. *Communications of the ACM*, 33(8):30–49.

[108] Li, Y. (2017). Deep Reinforcement Learning: An Overview. In *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, pages 1–70. Springer.

[109] Li, Z., Cheng, G., and Qiang, X. (2010). Some Classical Constructive Neural Networks and their New Developments. In *ICENT*. IEEE.

[110] Li, Z. and Hoiem, D. (2018). Learning without Forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

[111] Lin, L. J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321.

[112] Lin, L.-J. and Mitchell, T. M. (1993). Reinforcement learning with hidden states. In Meyer, J.-A., Roitblat, H. L., and Wilson, S. W., editors, *From animals to animats 2*, pages 271–280. MIT Press.

[113] Lin, M., Chen, Q., and Yan, S. (2014). Network In Network. *arXiv preprint*, page 10.

[114] Lopez-Paz, D. and Ranzato, M. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS 2017)*, pages 6468–6477.

[115] Lu, P., Chen, S., and Zheng, Y. (2012). Artificial Intelligence in Civil Engineering. *Mathematical Problems in Engineering*, 12:1–22.

[116] Lv, K., Jiang, S., and Li, J. (2017). Learning Gradient Descent: Better Generalization and Longer Horizons. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2247–2255, Sydney, Australia.

[117] Martinez-hernandez, U. and Prescott, T. J. (2017). Adaptive perception : learning from sensory predictions to extract object shape with a biomimetic fingertip. In *IEEE / RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6735–6740, Vancouver, BC, CAnada.

[118] McCallum, A. K. (1995). *Reinforcement Learning with Selective Percemption and Hidden State*. PhD thesis, University of Rochester.

[119] McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. *AI Magazine*, 27(4):12–14.

[120] Mcculloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

[121] Milan, K., Veness, J., Kirkpatrick, J., Hassabis, D., Koop, A., and Bowling, M. (2016). The Forget-me-not Process. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, pages 3702–3710.

[122] Minsky, M. and Papert, S. (1969). *Perceptrons : an introduction to computational geometry*. MIT Press.

[123] Mitchell, B. R. (2017). *The spatial inductive bias of deep learning*. Phd thesis, John Hopkins University.

[124] Mitchell, T. M. (1980). The Need for Biases in Learning Generalizations. In *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann Publishers.

[125] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, New York, NY, USA.

[126] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[127] Mozer, M. C. (1989). A Focused Backpropagation Algorithm for Temporal Pattern Recognition. *Complex Systems*, 3:349–381.

[128] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., and Silver, D. (2015). Massively Parallel Methods for Deep Reinforcement Learning. *arXiv:1507.04296*, page 14.

[129] Natarajan, B. K. (1989). On Learning Sets and Functions. *Machine Learning*, 4(1):67–97.

[130] Nichol, A., Pfau, V., Hesse, C., Klimov, O., and Schulman, J. (2018). Gotta Learn Fast: A New Benchmark for Generalization in RL. *arXiv preprint*, pages 1–21.

[131] Nivel, E., Thorisson, K., Steunebrink, B., Dindo, H., Pezzulo, G., Rodriguez, M., Hernandez, C., Ognibene, D., Schmidhuber, J. H., Sanz, R., Helgason, H., Chella, A., and Jonsson, G. (2014). Autonomous acquisition of natural language. In *Proceedings of the 7th International Conference on Intelligent Systems & Agents*, pages 58–66, Lisbon, Portugal.

[132] Paul, S., Kurin, V., and Whiteson, S. (2019). Fast Efficient Hyperparameter Tuning for Policy Gradients. *arXiv preprint*, pages 1–16.

[133] Pavlov, I. P. (1927). *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex. Translated and Edited by G. V. Anrep.* Oxford University Press.

[134] Peshkin, L., Meuleau, N., and Kaelbling, L. (1999). Learning Policies with External Memory. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-1999)*, pages 307–314.

[135] Peters, J., Mülling, K., and Altün, Y. (2010). Relative Entropy Policy Search. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI-10)*, pages 1607–1612.

[136] Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

[137] Piaget, J. (1952). *Play, dreams and imitation in childhood.*, volume 16. W. W. Norton & Co., New York, NY, USA.

[138] PwC (2018). The macroeconomic impact of artificial intelligence. Technical report, PWC.

[139] Ring, M. B. (1994). *Continual learning in reinforcement environments.* Phd thesis, University of Texas at Austin.

[140] Ring, M. B. (1997). CHILD: A First Step Towards Continual Learning. *Machine Learning*, 28(1):77–104.

[141] Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory.

[142] Rosman, B., Hawasly, M., and Ramamoorthy, S. (2016). Bayesian policy reuse. *Machine Learning*, 104(1):99–127.

[143] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(5):534–536.

[144] Rummery, G. A. and Niranjan, M. (1994). Online Q-learning Using Connectionist Sytems. Technical report, Cambridge University Engineering Department.

[145] Russell, S. and Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence*, 49(1-3):361–395.

[146] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016a). Progressive Neural Networks. *CoRR. arXiv:1606.04671.*

[147] Rusu, A. A., Sergio, G., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016b). Policy distillation. In *International Conference on Learning Representations (ICLR 2016)*, pages 1–13.

[148] Sandini, G., Metta, G., and Vernon, D. (2007). The iCub Cognitive Humanoid Robot : An Open-System Research Platform for Enactive Cognition Enactive Cognition : Why Create a Cognitive Humanoid. pages 358–369.

[149] Schacter, D. L., Gilbert, D. T., and Wegner, D. M. (2011). *Psychology (2nd Edition)*. Worth, New York.

[150] Schäfer, A. M. and Zimmermann, H. G. (2007). Recurrent Neural Networks Are Universal Approximators. In Kollias, S., Stafylopatis, A., Duch, W., and Oja, E., editors, *16th International Conference on Artificial Neural Networks*, volume 17, pages 253–263, Athens, Greece. Springer.

[151] Schaul, T. and Com, D. G. (2015). Universal Value Function Approximators. In *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning*, pages 1312–1320, Lille, France.

[152] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. In *International Conference on Learning Representations (ICLR 2016)*, pages 1–23.

[153] Schmidhuber, J. H. (1991). Curious Model-Building Control Systems. In *IEEE International Joint Conference on Neural Networks*, volume 2, pages 1458–1463.

[154] Schmidhuber, J. H. (1993). A self-referential Weight Matrix. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 446–451. Springer.

[155] Schmidhuber, J. H. (1994). On Learning how to Learn Learning Strategies. Technical report, Fakultät für Informatik, Technische Universität München.

[156] Schmidhuber, J. H. (1995). Beyond Genetic Programming: incremental self-improvement. In Rosca, J., editor, *Proc. Workshop on Genetic Programming at ML95*, pages 42–49.

[157] Schmidhuber, J. H. (1999). A general method for incremental self-improvement and multi-agent learning. In Yao, X., editor, *Evolutionary Computation: Theory and Applications.*, volume 1, chapter 3, pages 81–123. World Scientific.

[158] Schmidhuber, J. H. (2002). Exploring the Predictable. In Ghosh, S. and Tsutsui, S., editors, *Advances in Evolutionary Computing*, pages 579–612. Springer.

[159] Schmidhuber, J. H. (2004). Optimal Ordered Problem Solver. *Machine Learning*, 54:211–254.

[160] Schmidhuber, J. H. (2005). Gödel Machines : Towards a Technical Justification of Consciousness. In Kudenko, D., Kazakov, D., and Alonso, E., editors, *Adaptive Agents and Multi-Agent Systems II*, pages 1–23. Springer.

[161] Schmidhuber, J. H. (2006). Gödel Machines : Self-Referential Universal Problem Solvers Making Provably Optimal Self-Improvements. Technical report, IDSIA, Galleria 2, 6928 Manno (Lugano), Switzerland.

[162] Schmidhuber, J. H. (2007). Gödel Machines : Fully Self-referential Optimal Universal Self-improvers. In Goertzel, B. and Pennachin, C., editors, *Artificial General Intelligence*, pages 199–226. Springer.

[163] Schmidhuber, J. H. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61:85–117.

[164] Schmidhuber, J. H. and Zhao, J. (1997). Multi-agent learning with the success-story algorithm. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1221 LNAI:82–93.

[165] Schmidhuber, J. H., Zhao, J., and Wiering, M. (1996). Simple principles of metalearning. Technical report, IDSIA, Lugano, Switzerland.

[166] Schmidhuber, J. H., Zhao, J., and Wiering, M. (1997). Shifting Inductive Bias with Success-Story Algorithm, Adaptive Levin Search, and Incremental Self-Improvement. *Machine Learning*, 28:105–130.

[167] Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. (2015). Trust Region Policy Optimization. In *Proceedings of the 31st International Conference on Machine Learning*, pages 205–212, Lille, France.

[168] Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional Continuous Control Using Generalised Advantage Estimation. *ICLR 2016*.

[169] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint*, pages 1–12.

[170] Schulze, C. and Schulze, M. (2018). ViZDoom: DRQN with Prioritized Experience Replay, Double-Q Learning, & Snapshot Ensembling. In *Proceedings of SAI Intelligent Systems Conference*, pages 1–17.

[171] Serrà, J., Surís, D., Miron, M., and Karatzoglou, A. (2018). Overcoming catastrophic forgetting with hard attention to the task. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, Stockholm, Sweden. PMLR.

[172] Shani, G., Pineau, J., and Kaplow, R. (2013). A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems*, 27(1):1–51.

[173] Shapiro, S. C. and Rapaport, W. J. (1992). The SNePS family. *Computers & Mathematics with Applications*, 23:243–275.

[174] Shibata, K. (2017). Functions that Emerge through End-to-End Reinforcement Learning - The Direction for Artificial General Intelligence. *arXiv preprint*, pages 1–4.

[175] Shibata, K. and Goto, K. (2013). Emergence of flexible prediction-based discrete decision making and continuous motion generation through actor-Q-learning. *2013 IEEE 3rd Joint International Conference on Development and Learning and Epigenetic Robotics, ICDL 2013 - Electronic Conference Proceedings*, pages 2–7.

[176] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D., Turing, A., and Shannon, C. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144.

[177] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. In *Proceedings of the 31 st International Conference on Machine Learning*, Bejing, China.

[178] Silver, D., Sutton, R. S., and Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine learning*, pages 968–975, Helsinki, Finland.

[179] Silver, D. and Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. *Advances in neural information processing systems (NIPS).*, pages 1–9.

[180] Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong Machine Learning Systems : Beyond Learning Algorithms. *2013 AAAI Spring Symposium Series*, pages 49–55.

[181] Singh, S. P., Barto, A. G., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems 17 (NIPS 2004)*, pages 1281–1288.

[182] Skinner, B. (1938). *The Behavior of Organisms: An Experimental Analysis.* B.F. Skinner Foundation, Cambridge, Massachusetts.

[183] Sontag, E. D. (1998). VC dimension of neural networks. *NATO ASI Series F Computer and Systems Sciences*, 168:69–96.

[184] Sparkes, A., Aubrey, W., Byrne, E., Clare, A., Khan, M. N., Liakata, M., Markham, M., Rowland, J., Soldatova, L. N., Whelan, K. E., Young, M., and King, R. D. (2010). Towards Robot Scientists for autonomous scientific discovery. *Automated Experimentation*, 2(1):1–11.

[185] Spearman, C. (1904). General Intelligence , Objectively Determined and Measured. *The American Journal of Psychology*, 15(2):201–292.

[186] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, pages 1–14.

[187] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212.

[188] Stanley, K. O. and Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, 10(2):99–127.

[189] Steels, L. (2008). The Symbol Grounding Problem has been solved. So what's next? In de Vega, M., Glenberg, A. M., and Graesser, A. C., editors, *Symbols and Embodiment: Debates on Meaning and Cognition*, pages 223–244. Oxford University Press.

[190] Sternberg, R. (2000). The Theory of Successful Intelligence. *Gifted Education International*, 15:4–21.

[191] Steunebrink, B. R. and Schmidhuber, J. (2011). A family of Gödel machine implementations. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6830 LNAI:275–280.

[192] Steunebrink, B. R. and Schmidhuber, J. H. (2012). Towards An Actual Goedel Machine Implementation : A Lesson in Self-Reflective Systems 1. In Wang, P. and Goertzel, B., editors, *Theoretical Foundations of Artificial General Intelligence*, pages 173–196. Springer.

[193] Steunebrink, B. R., Thórisson, K. R., and Schmidhuber, J. H. (2016). Growing Recursive Self-Improvers. In *International Conference on Artificial General Intelligence (AGI-2016)*, pages 129–139.

[194] Stevan Harnad (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346.

[195] Storck, J., Hochreiter, S., and Schmidhuber, J. H. (1995). Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International Conference on Artificial Neural Networks (ICANN95)*, 2:159–164.

[196] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J. (2017). Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv preprint*.

[197] Sun, R. and Zhang, X. (2004). Top-down versus bottom-up learning in cognitive skill acquisition. *Cognitive Systems Research*, 5:63–89.

[198] Sung, F., Zhang, L., Xiang, T., Hospedales, T., and Yang, Y. (2017). Learning to Learn: Meta-Critic Networks for Sample Efficient Learning. *arXiv preprint*, pages 1–12.

[199] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112.

[200] Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., and White, A. (2011). Horde : A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction Categories and Subject Descriptors. In Tumer, Yolum, S. and Stone, editors, *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, volume 2, pages 761–768, Taipei, Taiwan.

[201] Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211.

[202] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

[203] Szita, I. and Lorincz, A. (2006). Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, 18:2936–2941.

[204] Taddeo, M. and Floridi, L. (2005). Solving the Symbol Grounding Problem: a Critical Review of Fifteen Years of Research. *Artificial Intelligence*, 3079(December):17–19.

[205] Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains : A Survey. *Journal of Machine Learning Research*, 10:1633–1685.

[206] Tesauro, G. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–88.

[207] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016). A Deep Hierarchical Approach to Lifelong Learning in Minecraft. *arXiv*, pages 1–6.

[208] Thorndike, E. L. (1898). Animal Intelligence: An Experimental Study of the Associate Processes in Animals. *Psychological Review Monograph Supplement*, 2(4):1–8.

[209] Thrun, S. and Schwartz, A. (1995). Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 385–392.

[210] Thrun, S. and Sullivan, J. O. (1995). Clustering Learning Tasks and the Selective Cross-Task Transfer of Knowledge. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA.

[211] Thrun, S. B. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15:25–46.

[212] Turing, A. M. (1938). On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, s2-43(1):544–546.

[213] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 49:433–460.

[214] UK Space Agency (2018). UK space sector set to benefit from new European Space Agency contract. Retrieved at 27 October from https://www.gov.uk/government/news/uk-space-sector-set-to-benefit-from-new-european-space-agency-contract.

[215] University of Southampton (2017). The Iridis Compute Cluster.

[216] Urban, J. (2008). Automated reasoning for mizar: Artificial intelligence through knowledge exchange. *CEUR Workshop Proceedings*, 418:1–16.

[217] Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

[218] Vamplew, P. and Ollington, R. (2005). On-Line Reinforcement Learning Using Cascade Constructive Neural Networks. In Khosla, R., Howlett, . J., and Jain, . C., editors, *9th International Conference, KES 2005*, pages 562–568. Springer.

[219] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. In *AAAI'16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100.

[220] Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of probability and its applications*, 16(2):264–280.

[221] Vargha, A. and Delaney, H. D. (2000). A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132.

[222] Veness, J., Ng, K. S., Hutter, M., Uther, W., and Silver, D. (2011). A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142.

[223] Wang, P. (2007). The logic of intelligence. In *Artificial General Intelligence*, pages 31–62. Springer.

[224] Wang, Z., Schaul, T., Hessel, M., and Lanctot, M. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, New York, NY, USA. JMLR: W&CP.

[225] Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* PhD thesis, King's College.

[226] Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

[227] Wayne, G., Graves, A., Uria, B., Kalchbrenner, N., and Graves, A. (2016). Associative Long Short-Term Memory. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, New York, NY, USA.

[228] Wierstra, D., Forster, A., Peters, J., and Schmidhuber, J. H. (2010). Recurrent Policy Gradients. *Logic Journal of IGPL*, 18(5):620–634.

[229] Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256.

[230] Williams, R. J. and Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2):270–280.

[231] Wilson, A., Fern, A., Ray, S., and Tadepalli, P. (2007). Multi-task reinforcement learning: a hierarchical Bayesian approach. *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022.

[232] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The Marginal Value of Adaptive Gradient Methods in Machine Learning. In *31st Conference on Neural Information Processing Systems (NIPS 2017)*, pages 4148–4158, Long Beach, CA, USA.

[233] Wolpert, D. H. (1996). The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390.

[234] Wu, T. and Tegmark, M. (2019). Toward an artificial intelligence physicist for unsupervised learning. *Physical Review E*, 100(3):33311.

[235] Xu, T., Liu, Q., Zhao, L., and Peng, J. (2018). Learning to Explore via Meta-Policy Gradient. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden.

[236] Yampolskiy, R. V. (2015). Analysis of Types of Self-Improving Software. In Bieger, J., Goertzel, B., and Potapov, A., editors, *International Conference on Artificial General Intelligence (AGI-2015)*, pages 384–393. Springer.

[237] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS 2014)*, 27:1–9.

[238] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *arXiv preprint*, pages 1–6.

[239] Zeng, S. Q., Tham, K. Y., Badgero, M., and Weng, J. Y. (2002). Dav : A Humanoid Robot Platform for Autonomous Mental Development. In *Proceedings of the 2nd IEEE conference on Development and Learning*, pages 73–81.

[240] Zhao, J. (2002). Self-modifying Reinforcement Learning. In *Proceedings of the First International Conference on Machine Learning and Cybernetics*, pages 1–38, Beijing, China.

[241] Zhao, J. and Schmidhuber, J. H. (1998). Solving a Complex Prisoner's Dilemma with Self-Modifying Policies. In *From Animals to Animats 5: Proc. 5th International Conference on Simulation of Adaptive Behavior*, pages 177–182, Zurich, Switzerland.