# Learning to learn with active adaptive perception

**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.

*Keywords:* adaptive perception, inductive bias, self-modifying policies, reinforcement learning, partial observability

## 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 et al., 2017; Li, 2017; Schmidhuber, 2015; Lecun et al., 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 modeling 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 (Whitehead & Ballard, 1990; Crook, 2006) or for repositioning sensors to better recognise patterns in the environment Shibata et al. (2001); Mnih et al. (2014). 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 (Sorokin et al., 2015; Hausknecht & Stone, 2015; Bakker, 2002; Wierstra et al., 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 learn. The generality of such approaches is reduced due to the inherent inductive bias in a single learning algorithm (Wolpert & Macready, 1997; Mitchell, 1980). To address the limits of specialised AI approaches, research into Artificial General Intelligence (AGI) or Human-level artificial intelligence (Nilsson, 2005; McCarthy, 2007; Adams et al., 2012; Baum et al., 2011) is making efforts towards learners with a generality greater or equal to humans. To achieve AGI, various different approaches have been proposed. Symbolic approaches (Laird, 2012; Shapiro & Rapaport, 1992; Choi & Langley, 2018; Lenat et al., 1990; Kieras & Meyer, 1997; Anderson et al., 2004) 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 et al., 2010), while others use a developmental robotics (Asada et al., 2009; Zeng et al., 2002; Sandini et al., 2007) approach. The latter are heavily inspired by reinforcement learning with intrinsic motivation (Storck et al., 1995; Schmidhuber, 1991; Singh et al., 2004), lifelong reinforcement learning (Bengio et al., 2009; Thrun & Mitchell, 1995; Silver et al., 2013; Ring, 1994), 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; Franklin et al., 2014; Cassimatis et al., 2004; Goertzel et al., 2011, 2013; 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, 2007, 2004; 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 (Schmidhuber et al., 1997a; Schmidhuber, 2007; Orseau & Ring, 2011; Everitt et al., 2016). 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 occurs, 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 & Zhao, 1997; Zhao & Schmidhuber, 1998; Schmidhuber, 1999), noisy environments (Zhao, 2002), as well as continual learning, solving problems of increasing complexity presented sequentially across the lifetime (Schmidhuber et al., 1997b).

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 et al., 1997a; Schmidhuber & Zhao, 1997; Zhao & 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; Schmidhuber et al., 1996; Schmidhuber, 1999) 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 et al. (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 et al., 1998; Silver & Veness, 2010; Shani et al., 2013) , 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

[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 Vonn 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.

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.

## 3. Generic architecture for active adaptive perception

This section proposes a generic architecture and an exemplar implementation, illustrated in Figure 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 $\mathcal{P} : \mathcal{O} \to \mathcal{A}$ 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 $\mathcal{A}$. 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 $a_1, \ldots, a_N$, 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 continously 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 $\mathcal{P}_i$ ($i = ProgramStart, \ldots, ProgramStart+m-1$) 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 $IP$, 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 3.6, while a complete instruction set is given in Table 2.

### *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 $t_1$ is 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 $[t_1, t_2]$ 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 Figure 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:

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 2, 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 $\ell$, 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 $\ell$.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 $weight$ 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 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

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

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); $\gamma$ is the discount used to compute the discounted future cumulative reward $\sum_{i=0}^{\infty} \gamma^i r^i$; $\theta$ 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 $\epsilon$-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 $\epsilon$ 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 $\epsilon$. **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 $a_1$, $\text{E}_{a_1}$, 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 examplary 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 $13 \times 13$. 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 hours 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 Figure 3, illustrating that methods of active adaptive perception avoid misleading corridors and rooms more often than other methods. Figure 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 3. 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 Figure 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 Figure 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 Figure 3, where the DRQN got stuck indefinitely. This is because DRQN remains inside a detracting room or

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

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 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 Figure 6 and Figure 7. 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 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 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 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 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. Figure 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 Figure 5b, due to the $\mathcal{P}$-modifications, which increase the *until* parameter to allow itself more time to reach the more difficult goals. Figure 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 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 4c, where it can be observed that detracting corridors have relatively high exploration rates compared to DRQN.

*Average performance.* The development plots in Figures 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 4, 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 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 5. 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 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 et al., 1997b) 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 (Sharma & Chandra, 2010; Vamplew & Ollington, 2005; Lahnajarvi et al., 2002; Fanguy & Kubat, 2002; Parekh et al., 2000; Fahlman & Lebiere, 1990; Frean, 1990; Ring, 1997) the architecture 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 et al., 1997b).

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 et al., 1997a) , 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 et al., 1994; Hochreiter & Schmidhuber, 1997), comparable issues may occur due to the exploding gradient (Sutskever et al., 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 et al., 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 (Yang et al., 2016; Mahadevan, 1996) 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 parallellism 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 et al., 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 lead 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.

## 8. Acknowledgements

This research was funded by 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.

## References

Adams, S., Arel, I., Bach, J., Coop, R., Furlan, R., Goertzel, B., Hall, J. S., Samsonovich, A., Scheutz, M., Schlesinger, M., Shapiro, S. C., & Sowa, J. (2012). Mapping the Landscape of Human-Level Artificial General Intelligence. *AI Magazine*, *33*, 25–42. URL: `http://www.aaai.org/ojs/index.php/aimagazine/article/view/2322`. doi:`10.1609/aimag.v33i1.2322`.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An Integrated Theory of the Mind. *Psychological Review*, *111*, 1036–1060. doi:`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., Ogino, M., & Yoshida, C. (2009). Cognitive Developmental Robotics: A Survey. *IEEE Transactions on Autonomous Mental Development*, *1*, 12–34. doi:10.1109/TAMD.2009.2021702.

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

Bajcsy, R., Aloimonos, Y., & Tsotsos, J. K. (2017). Revisiting active perception. *Autonomous Robots*, . doi: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*, 185–195.

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*. Barcelona, Spain. URL: http://arxiv.org/abs/1606.01868. doi:10.2172/1336367. 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*, 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*, 13–23. doi:10.1016/j.robot.2004.07.014.

Choi, D., & Langley, P. (2018). Evolution of the ICARUS Cognitive Architecture. *Cognitive Systems Research*, *48*, 25–38. doi: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*, 19–41.

Frean, M. (1990). The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks. *Neural Computation*, *2*, 198–209. doi: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., Watkins, O., & Yu, G. (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., Huang, D., & Yu, G. (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 Cognitive* (pp. 1436–1441).

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, *21*, 178–192. doi: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*, 1–32. doi: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*, 119–131. doi: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*, 99–134. URL: http://linkinghub.elsevier.com/retrieve/pii/S000437029800023X. doi:10.1016/S0004-3702(98)00023-X. 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). URL: http://ieeexplore.ieee.org/document/5708961/. doi:10.1109/ICMLA.2010.138.

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*. doi: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. doi: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*, 159–196. URL: http://link.springer.com/10.1023/A:1018064306595. doi:10.1023/A:1018064306595.

McCarthy, J. (2007). From here to human-level AI. *Artificial Intelligence*, *171*, 1174–1182. doi: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., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning*. New York, NY, USA volume 48. URL: http://arxiv.org/abs/1602.01783. doi:10.1177/0956797613514093. arXiv:1602.01783.

Mnih, V., Hess, N., Graves, A., & Kavukcuoglu, K. (2014). Recurrent Models of Visual Attention. *Proceedings of the 27th International Conference on Neural Information Processing Systems*, *2*, 2204–2212. doi:10.1017/S037346330300239X. arXiv:1406.6247v1.

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., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–540. doi:10.1038/nature14236. arXiv:1604.03986.

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

Nivel, E., Thorisson, K. R., Steunebrink, B. R., Dindo, H., Pezzulo, G., Rodriguez, M., Hernandez, C., Ognibene, D., Schmidhuber, J., Sanz, R., Helgason, H. P., & Chella, A. (2014). Bounded seed-AGI. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8598 LNAI*, 85–96. doi:10.1007/978-3-319-09274-4_9.

Nivel, E., Thórisson, K. R., Steunebrink, B. R., Dindo, H., Pezzulo, G., Rodriguez, M., Hernandez, C., Ognibene, D., Schmidhuber, J., Sanz, R., Helgason, H. P., Chella, A., & Jonsson, G. K. (2013). *Bounded Recursive Self-Improvement*. Technical Report December Reykjavik University. arXiv:1312.6764.

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*, 436–451. doi:10.1109/72.839013.

Piaget, J. (1952). Play, dreams and imitation in childhood. doi: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*, 77–104. doi: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. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, *61*, 85–117. doi:10.1016/j.neunet.2014.09.003. arXiv:1404.7828.

Schmidhuber, J. H. (1991). Curious Model-Building Control Systems. In *IEEE International Joint Conference on Neural Networks* (pp. 1458–1463). volume 2. doi: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.* chapter 3. (pp. 81–123). World Scientific volume 1.

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. H., & 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. doi:10.1007/3-540-62934-3_43.

Schmidhuber, J. H., Zhao, J., & Schraudolph, N. N. (1997a). 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*. Technical Report IDSIA Lugano.

Schmidhuber, J. H., Zhao, J., & Wiering, M. (1997b). Shifting Inductive Bias with Success-Story Algorithm, Adaptive Levin Search, and Incremental Self-Improvement. *Machine Learning*, *28*, 105–130. doi: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*, 1–51. doi:10.1007/s10458-012-9200-2.

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

Sharma, S. K., & Chandra, P. (2010). Constructive neural networks: a review. *International journal of engineering science and technology*, *2*, 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. *2013 IEEE 3rd Joint International Conference on Development and Learning and Epigenetic Robotics, ICDL 2013 - Electronic Conference Proceedings*, (pp. 2–7). doi: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* (pp. 1000–1005). Seoul, Korea volume 1. doi:10.1109/ROBOT.2001.932680.

Silver, D., & Veness, J. (2010). Monte-Carlo Planning in Large POMDPs. *Advances in neural information processing systems (NIPS).*, (pp. 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. *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)*, (pp. 1281–1288).

Sorokin, I., Seleznev, A., Pavlov, M., Fedorov, A., & Ignateva, A. (2015). Deep Attention Recurrent

Q-Network. arXiv:1512.01693.
Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, *10*, 99–127. doi:10.1162/106365602320169811.
Storck, J., Hochreiter, S., & Schmidhuber, J. (1995). Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International Conference on Artificial Neural Networks (ICANN95)*, *2*, 159–164.
Sun, R., & Zhang, X. (2004). Top-down versus bottom-up learning in cognitive skill acquisition. *Cognitive Systems Research*, *5*, 63–89. doi: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)*, (pp. 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, . J. Howlett, & . 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*, 279–292. URL: http://link.springer.com/10.1007/BF00992698. doi: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*, 620–634.
Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*, 67–82. doi: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 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.

$sizeX \leftarrow 13$; $sizeY = 13$;
$compl \leftarrow .10$; $density \leftarrow .10$;
$length \leftarrow floor(complexity * (5 * (sizeX + sizeY)))$;
$islands \leftarrow floor(density * ((sizeX/2) * (sizeY/2)))$;
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`$((y, x - 2)$;
        **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**
            $\tilde{y}, \tilde{x} \leftarrow$ `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 $(\bar{y}, \bar{x})$;
                $(y, x) \leftarrow (\tilde{y}, \tilde{x})$
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`$(p)$;

## 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 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: $Max = \mathcal{A}+m$; $Min = Max-N_{wm}$; $RegisterStart = 0$; $InputEnd = Min+8$. The input cells had addresses in $Min, \ldots, InputEnd$, the working cells had addesses 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 6, were the same as in (Hausknecht & 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 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.

| **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** |

**Table 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.*

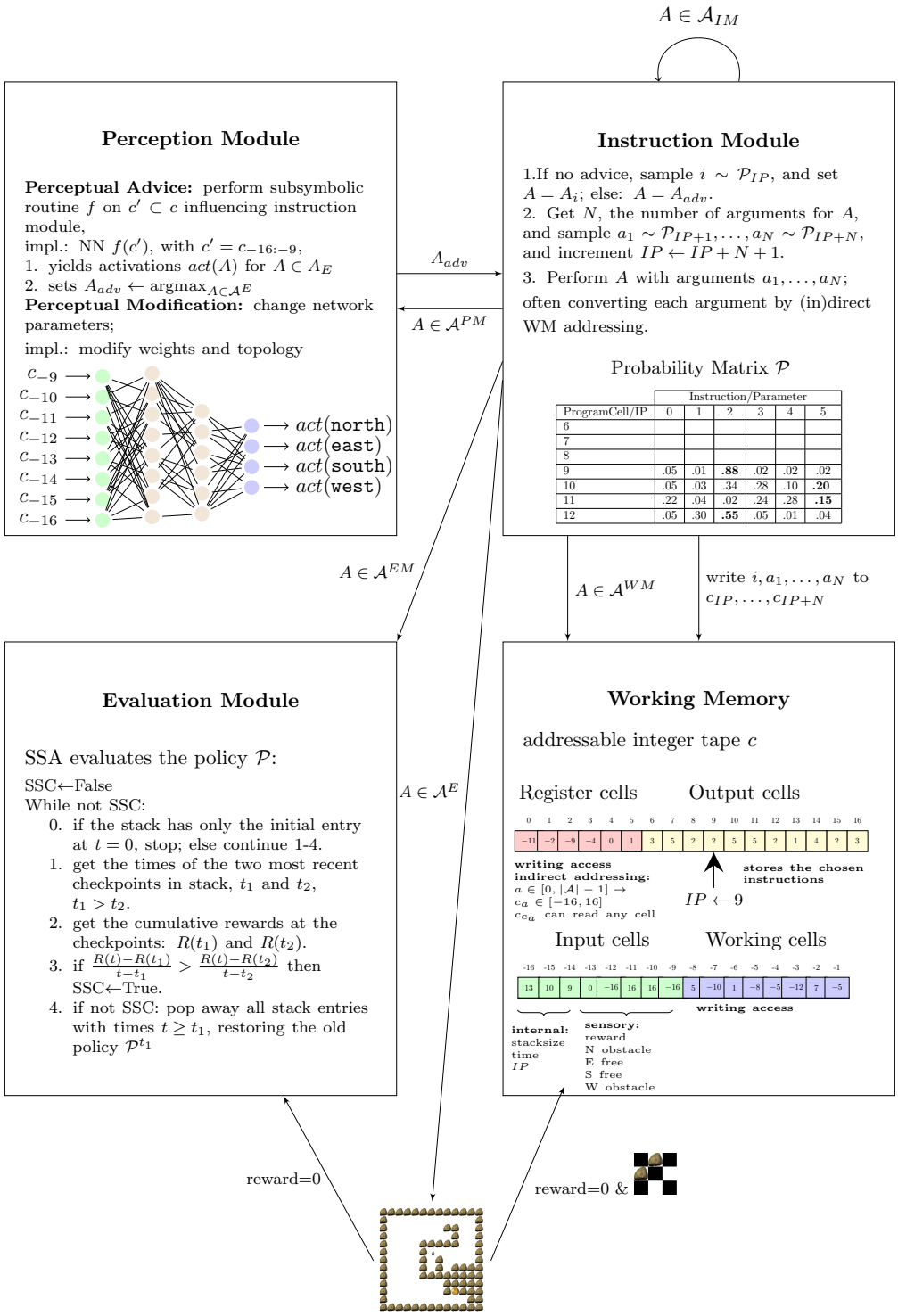
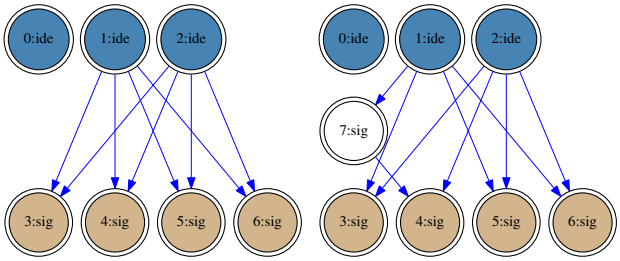


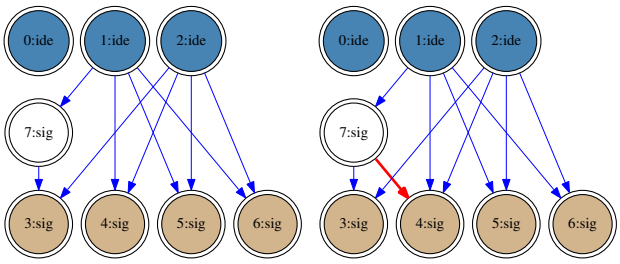

**Figure 1:** *Diagram of the generic architecture for active adaptive perception, and its current implementation on the maze problem. Based on the current instruction pointer as an internal state, the instruction module generates an instruction and its arguments. Working memory elements, integers in* $[-16, 16]$*, 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.*

| 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 $\epsilon$-greedy strategy, with the self-chosen $\epsilon$, 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 $\text{E}_{a_1}$ 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 $c_{a_1} \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\}$ ; $\mathcal{P}_{c_{a_1},i} \leftarrow .01 * c_{c_{a_3}} * \mathcal{P}$ for all $i \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\}$ |
| `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 $c_{a_1} \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\}$; $\mathcal{P}_{c_{a_1},i} \leftarrow \mathcal{P}_{c_{a_1},i} * (1 - .01 * c_{c_{a_3}} * \mathcal{P}_{c_{a_1},c_{a_2}})/(1 - \mathcal{P}_{c_{a_1},c_{a_2}})$ for all $i \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\} \setminus c_{a_2}$. Reject the modification if $\mathcal{P}_{c_{a_1},i} < minP = 0.0005$ for any $i \in \{0, \ldots, \lvert\mathcal{A}\rvert - 1\}$. |
| `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 2:** *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 $a \in [0, \lvert\mathcal{A}\rvert - 1]$ 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. (Schmidhuber, 1999)); if these conditions are not met the operation does nothing except for the usual increments to the instruction pointer $IP$.*




**(a)** *addNode*

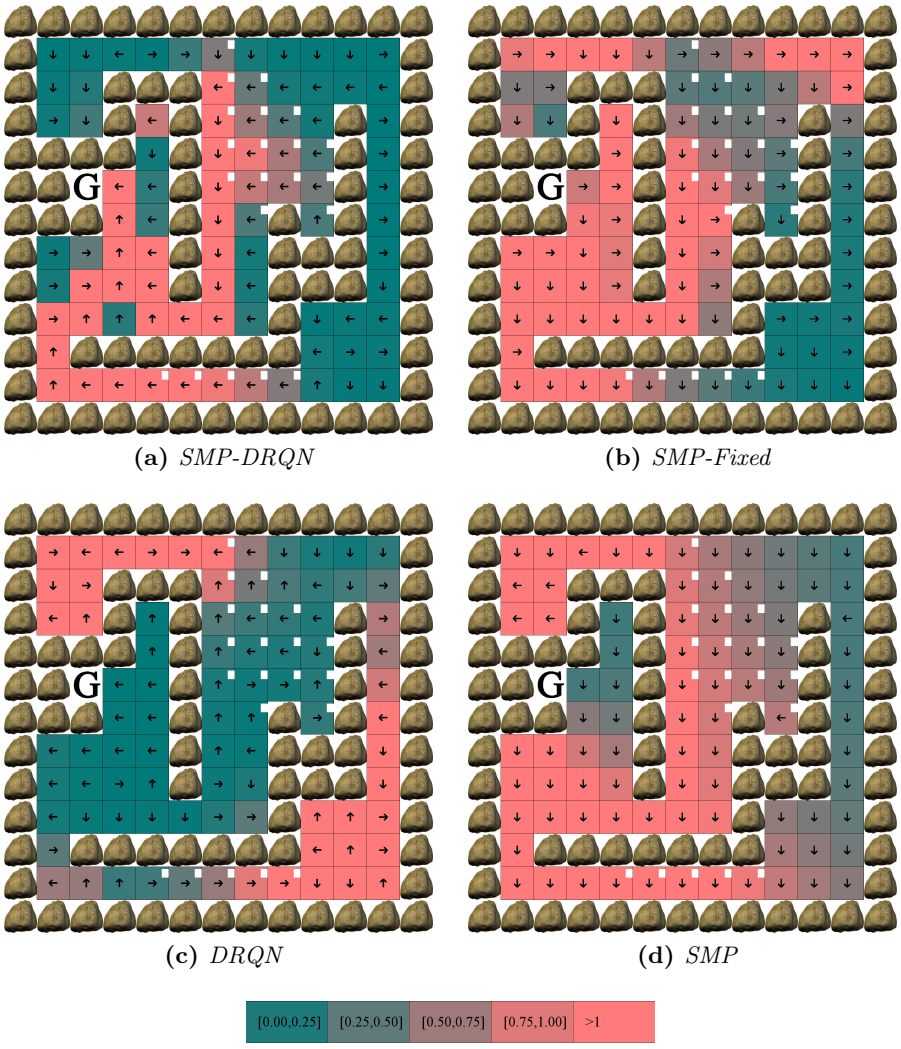



**(b)** *addConnection*

**Figure 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.*

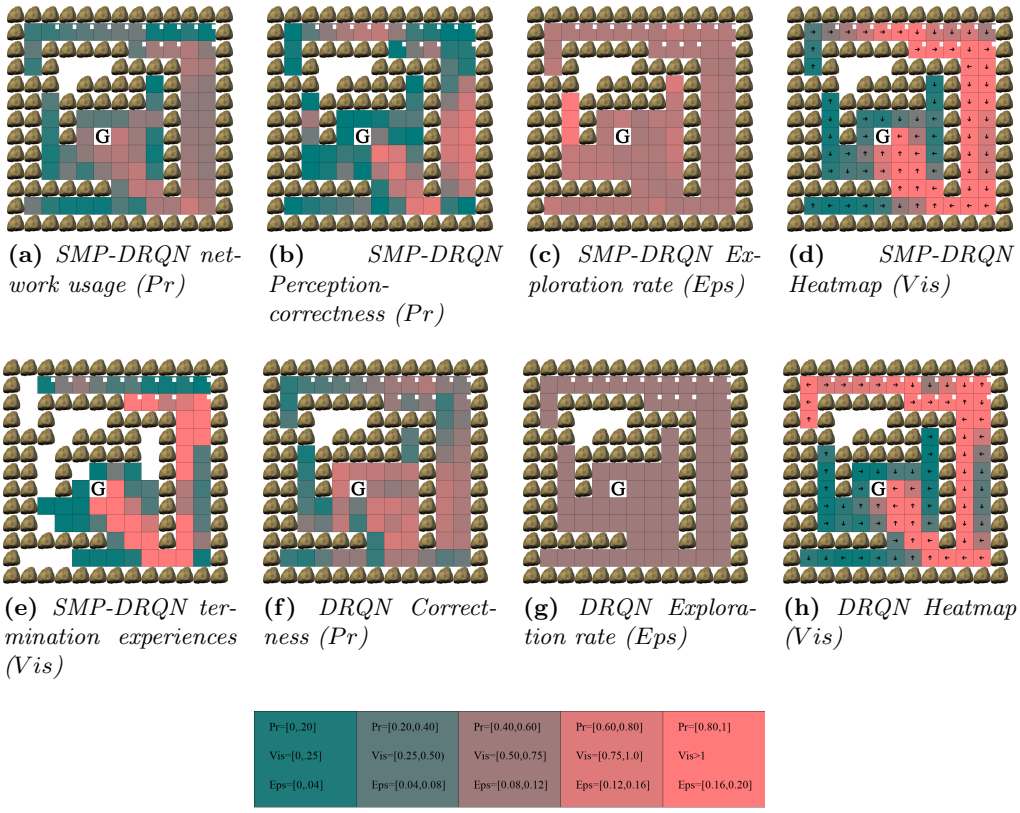
| | Environment | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Easy-Fixed | | | Easy-Random | | | Difficult-Fixed | | | Difficult-Random | | |
| | $C_i$ | $C_f$ | $r$ | $C_i$ | $C_f$ | $r$ | $C_i$ | $C_f$ | $r$ | $C_i$ | $C_f$ | $r$ |
| DRQN | .48 | .79 | .75 | .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 |

**Table 3:** *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.*




**(a)** *SMP-DRQN* **(b)** *SMP-Fixed* **(c)** *DRQN* **(d)** *SMP*

**Figure 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).*




**(a)** *SMP-DRQN network usage (Pr)*

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

**(c)** *SMP-DRQN Exploration rate (Eps)*

**(d)** *SMP-DRQN Heatmap (Vis)*

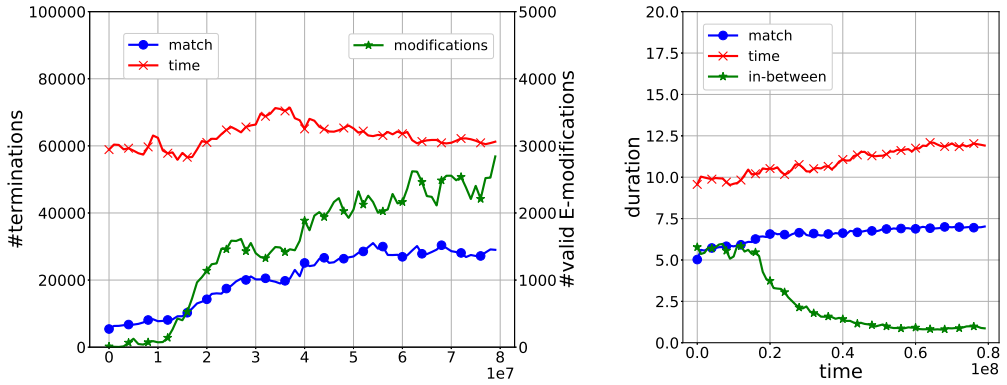
**(e)** *SMP-DRQN termination experiences (Vis)*

**(f)** *DRQN Correctness (Pr)*

**(g)** *DRQN Exploration rate (Eps)*

**(h)** *DRQN Heatmap (Vis)*

**Figure 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 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 color of the plots is variable across figures, and their units are mentioned in parentheses; Pr is the proportion, Eps is the* $\epsilon$ *parameter for the* $\epsilon$*-greedy action selection, and Vis is the visitations per time unit times the number of unique visited locations.*

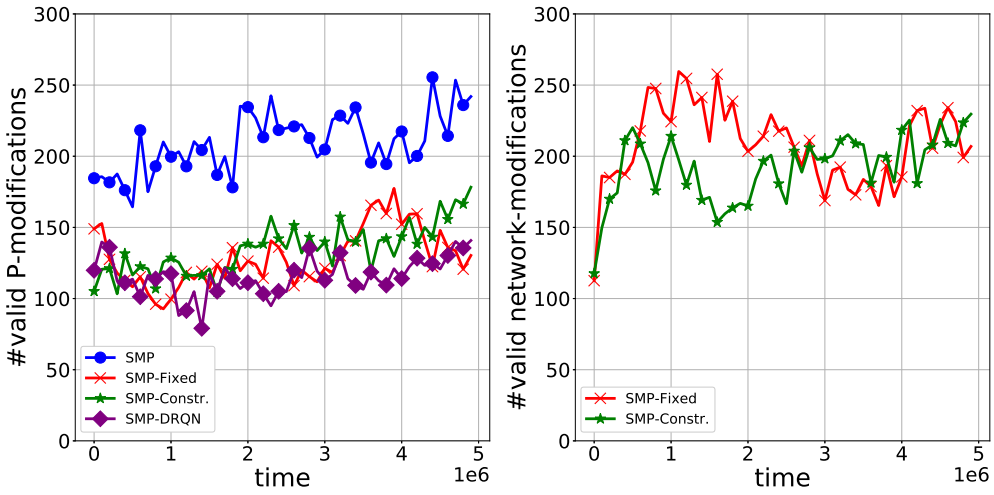



**(a)** *Match* **(b)** *Duration*

**Figure 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.*

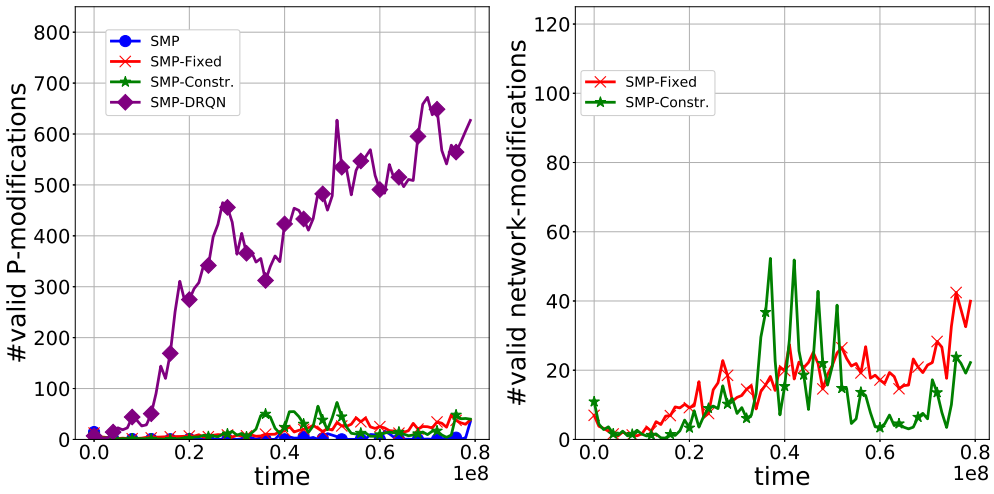
| | **Method** | **Performance** | **Comparison** | | | | |
|---|---|---|---|---|---|---|---|
| | | | DRQN | SMP-Constr. | SMP-Fixed | SMP | Random |
| Easy-Fixed | SMP-DRQN | $0.592 \pm 0.132$ | $< p = 0.644$ | $> p < 0.001$ | $> p < 0.001$ | $> p < 0.001$ | $> p < 0.001$ |
| | DRQN | $0.615 \pm 0.069$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

**Table 4:** *Life-time averaged normalised reward speed.* $<$ *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.*



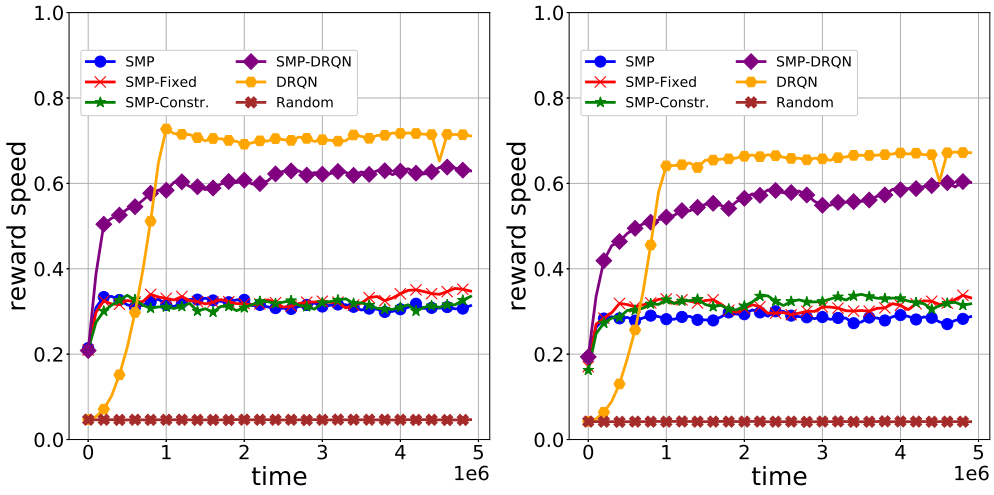

**(a)** *$\mathcal{P}$-modifications* **(b)** *network-modifications*

**Figure 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.*



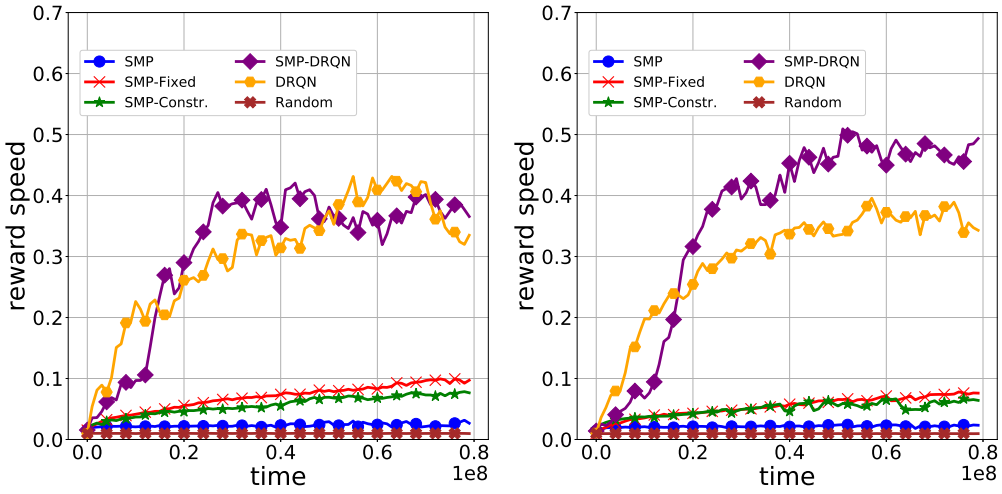

**(a)** *$\mathcal{P}$-modifications* **(b)** *network-modifications*

**Figure 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.*




**(a)** *Fixed* **(b)** *Random*

**Figure 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.*




**(a)** *Fixed* **(b)** *Random*

**Figure 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.

| | Fixed | | | Random | | |
|---|---|---|---|---|---|---|
| | rank | ratio | stuck | rank | ratio | stuck |
| DRQN | $1.2 \pm 0.4$ | $.97 \pm .08$ | $0.0 \pm .0$ | $1.4 \pm 0.5$ | $.97 \pm .06$ | $0.0 \pm .0$ |
| Random | $6.0 \pm 0.0$ | $.07 \pm .03$ | $0.0 \pm .0$ | $6.0 \pm 0.0$ | $.07 \pm .03$ | $0.0 \pm .0$ |
| SMP | $3.9 \pm 0.8$ | $.49 \pm .16$ | $0.0 \pm .0$ | $4.3 \pm 0.8$ | $.46 \pm .16$ | $0.0 \pm .0$ |
| SMP-Fixed | $3.9 + 0.8$ | $.50 \pm .16$ | $0.0 \pm .0$ | $3.7 \pm 0.8$ | $.51 \pm .18$ | $0.0 \pm .0$ |
| SMP-Constructive | $4.2 \pm 0.8$ | $.48 \pm .16$ | $0.0 \pm .0$ | $3.8 \pm 0.8$ | $.52 \pm .21$ | $0.0 \pm .0$ |
| SMP-DRQN | $1.7 + 0.4$ | $.91 \pm .12$ | $0.0 \pm .0$ | $1.7 \pm 0.7$ | $.90 \pm .14$ | $0.0 \pm .0$ |

**(a)** *Easy*

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

**(b)** *Difficult*

**Table 5:** *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* 10000 *time steps.*

| **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 (Zeiler, 2012) |
| learning rate | 0.1 |
| momentum | 0.95 |
| clip gradient | absolute value exceeding 10 |
| replay start | 50000 time steps |
| update frequency | 4 time steps |
| target update frequency | 10000 time steps |

**Table 6:** *DRQN parameters.*