To appear in *Neural Computation*, doi:10.1162/NECO_a_00514
(accepted in June 2013, in press)

Toward Nonlinear Local Reinforcement Learning Rules through Neuroevolution

Vassilis Vassiliades
v.vassiliades@cs.ucy.ac.cy
Chris Christodoulou
cchrist@cs.ucy.ac.cy
Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus

Keywords: reinforcement learning, learning rules, neural networks, neuroevolution

Abstract

We consider the problem of designing local reinforcement learning rules for artificial neural network (ANN) controllers. Motivated by the universal approximation properties of ANNs, we adopt an ANN representation for the learning rules, which are optimized using evolutionary algorithms. We evaluate the ANN rules in partially observable versions of four tasks: the mountain car, the acrobot, the cart pole balancing, and the nonstationary mountain car. For testing whether such evolved ANN-based learning rules perform satisfactorily, we compare their performance with the performance of SARSA(λ) with tile coding, when the latter is provided with either full or partial state information. The comparison shows that the evolved rules perform much better than SARSA(λ) with partial state information and are comparable to the one with

full state information, while in the case of the nonstationary environment, the evolved rule is much more adaptive. It is therefore clear that the proposed approach can be particularly effective in both partially observable and nonstationary environments. Moreover, it could potentially be utilized towards creating more general rules that can be applied in multiple domains and transfer learning scenarios.

1 Introduction

A reinforcement learning (RL) agent can be seen as an entity that has two components: (i) the policy/controller, which is a mapping from past observations to actions, and (ii) the learning rule, which modifies the policy towards better performance. A way to address large or continuous state-action spaces that need generalization is by representing the policy using parametric approximators. Feedforward artificial neural networks (ANNs) are known to be universal function approximators (Hornik et al., 1989; Park & Sandberg, 1991), while recurrent ANNs are considered as universal approximators of dynamical systems and can potentially represent arbitrarily complex mappings (Funahashi & Nakamura, 1993). For this reason, a particular form of direct policy search known as neuroevolution (NE, Floreano et al., 2008), i.e., the evolution of ANNs, has demonstrated remarkable performance over the years, even though it does not explicitly address problems such as partial observability or the exploration/exploitation tradeoff (Gomez et al., 2008). For example, in a partially observable environment, NE can find near-optimal non-stationary policies in the form of recurrent ANNs. While recurrent connections can be used to create adaptive agents, adaptation can also be achieved with the evolution of plastic ANNs, where a learning rule changes the synaptic weights online (Risi et al., 2010).

In this note, we turn our focus from the policy to the learning rule itself considering the latter to be a *mapping* as well. This mapping can potentially take as input the observations, the actions, the rewards and the parameters of the policy, and outputs a modified policy with the goal to improve (or maintain) performance. Our major aim is to investigate the emergence of general and robust learning rule mappings that could potentially be used for various classes of environments or tasks (e.g., generalized environments; Whiteson et al., 2011), as well as transferring learning rules (as opposed

to knowledge transfer) between related and potentially unrelated domains. The motivation behind this goal comes from the observation that families of RL rules from the RL literature can be used in a variety of tasks. For example, Q-learning (Watkins & Dayan, 1992) is a well-known family of RL rules that has been used in many domains.

Before even trying to tackle such a difficult problem, we first need to test whether promising approaches can be effective in simpler problems. More importantly though, we need to identify the parts or aspects that make the problem vague and resolve any ambiguities that might arise from them. For this reason, we outline below several choices we made before developing the approach presented in this work. These choices (and consequently, this study) reflect the starting point from which we set out to explore the potential realization of our goal. Concretely, this study investigates an approach where:

1. The policy is represented only by (plastic) ANNs, due to their universal approximation properties. Alternative policy representations, whilst interesting, fall outside the scope of this work.

2. The rules are not only optimized towards specific policy ANN architectures (Yao, 1999), but they are also coupled with the initial state of the ANNs that they modify. An alternative would be to develop rules that are optimized towards any ANN (topology and weights); this could not be done, unless the rules are manually designed with some guarantee in mind or with some external knowledge injected into the problem.

3. The rules require only local synaptic information and as a result make only local changes. An alternative would be to create global rules; local rules, however, are more biologically plausible and could have the same expressive power with the added advantage of requiring less information.

4. The rules do not modify the structure of the policy ANN, but only make synaptic modifications (synaptic plasticity). An alternative would be to allow structural changes as well (structural plasticity). Algorithms that modify both the structure and the parameters of ANNs, e.g., cascade-correlation (Fahlman & Lebiere, 1990), could be viewed as offering both structural and synaptic plasticity to the

ANN. In this regard, even optimization algorithms could be viewed as learning rules, however, we prefer to make a distinction between learning rules and optimization algorithms.

5. The rules are optimized using gradient-free methods and more specifically, evolutionary algorithms. Other gradient-free methods could be used, while an alternative would be to use gradient-based methods. Using the latter, however, it is not clear to us at this stage how to estimate the gradient and subsequently derive a learning rule that trains the learning rule. For this, which could be the subject of a separate study, as it demands a further and a more extensive investigation, an approach could be to utilize a forward model (for the rule, instead of the policy) and develop a learning rule that maximizes the predictive information of the weight update process and not of the sensorimotor process (as in Ay et al., 2012).

6. Only the parameters of the rules are optimized, not their internal structure, i.e., the "equation" of the rule does not change during optimization. Optimizing both the structure and the parameters clearly offers a lot of flexibility to the models. A drawback, however, is an expanded search space. While the (continuous) parameter space is considered to be infinite, structure learning makes the space combinatorial. Moreover, if the structure is allowed to vary, adding neurons increases the number of parameters and consequently the dimensionality of the problem. Algorithms that modify both the structure and the parameters of ANNs using evolutionary algorithms are sometimes called Topology and Weight Evolving ANNs (TWEANNs, see for example Stanley & Miikkulainen, 2002), but such algorithms are not used in this study.

7. Different RL rules are evolved for different RL tasks, i.e., the rule is overfit to the task. This is a starting point for this work as mentioned above. The proposed approach needs to be evaluated in simpler problems before moving to more complicated ones.

The remainder of this paper is organized as follows. Section 2 presents our approach and Section 3 describes the experimental setup. The results are presented in Section 4 followed by a discussion and conclusions in Section 5.

2 Approach

In local learning rules, the update requires only local synaptic information, i.e., pre-synaptic activation, post-synaptic activation and the connection weight. Such rules have the advantages of potentially reduced complexity and biological plausibility. An example of the latter is spike-timing-dependent plasticity, which has already been considered as a form of temporal difference (TD) learning (Rao & Sejnowski, 2001). We are interested in creating local RL rules and not only correlation-based (Hebbian) ones (Kolodziejski et al., 2008; Wörgötter & Porr, 2005); for this reason, the update is required to include the reward signal as an extra input. We use an ANN-based representation for storing the learning rules, since ANNs are capable of representing complex mappings. Therefore, two ANNs are used, one for the policy and one for the learning rule.

2.1 Policy ANN

In order to keep the policy ANN simple, its architecture has no hidden nodes, but a fully-recurrent connectivity at the output layer (thus the output neurons can act like hidden neurons). Both feedforward and recurrent connections can be adapted by the learning rule network. The input vector encodes the observation of the agent and has a dimensionality equal to the number of observation variables. A bias unit is also used. The output vector appropriately encodes the action of the agent depending on the task. For example, in an one-dimensional discrete action space, the number of output neurons is equal to the number of actions, and the action that corresponds to the neuron with the highest activation is selected. The activation function used is the hyperbolic tangent. The weights of the policy network are all initialized to 1.0 and the purpose of the learning rule network is to modify them in such a way, so as to maximize the average return per episode. The weight update rule is $\theta_{t+1}^{(ij)} = \theta_t^{(ij)} + \Delta\theta_t^{(ij)}$, where $\theta_t^{(ij)}$ is the current value of the connection weight between neurons i and j, and $\Delta\theta_t^{(ij)}$ is the weight change calculated by the learning rule ANN.

2.2 Learning rule ANN

The learning rule ANN (which is different from the policy ANN) uses a bias unit as well and has one output neuron that is responsible for a synaptic change. The output neuron uses the sigmoid function $\frac{8x}{1+|x|}$, where x is the net input, which has values in the range (-8,8). This was decided after some preliminary experimentation which revealed that a faster adaptation could be achieved in this range than with the standard hyperbolic tangent function (in the range [-1,1]) or with a piecewise linear function. The learning rule ANN could have the form:

$$\Delta\theta_t^{(ij)} = \Phi(o_{t+1}^{(i)}, o_t^{(i)}, ..., o_0^{(i)}, o_{t+1}^{(j)}, o_t^{(j)}, ..., o_0^{(j)}, \theta_t^{(ij)}, \theta_{t-1}^{(ij)}, ..., \theta_0^{(ij)}, r_{t+1}, r_t, ..., r_1) \quad (1)$$

where Φ corresponds to the change in the synaptic (policy) weight θ_t^{ij} between two neurons i and j, t is the discrete time step between agent decisions (stimulus-response time scale), $o_t^{(i)}, ..., o_0^{(i)}$ and $o_t^{(j)}, ..., o_0^{(j)}$ are the histories of pre- and post-synaptic activations respectively (at times $t, t-1, ..., 0$), $\theta_t^{(ij)}, \theta_{t-1}^{(ij)}, ..., \theta_0^{(ij)}$ is the history of the policy weight values, $r_{t+1}, r_t, ..., r_1$ is the history of reward signals (that are scaled in the range [-1,1]), and finally, $o_{t+1}^{(i)}$ and $o_{t+1}^{(j)}$ are the resulting pre- and post-synaptic activations respectively, when the policy network is activated using as input the observation variables obtained from the new state (at time $t+1$) *before* a synaptic modification is made. The latter means that the rule could effectively do an one-step simulation by seeing how the activation levels of the policy neurons change when using the unmodified weights and the new observation as input. It could then utilize these new activation levels for the calculation of the weight updates. This concept is similar to how dynamic programming and TD-learning methods work (i.e., bootstrapping). Each term in Equation 1 corresponds to an input in the learning rule ANN. The tasks could have an infinite-horizon, therefore, there needs to be a bound to the number of inputs. An approach would be to utilize a moving window. However, since recurrent connections can theoretically create internal representations that encode information for arbitrary long sequences, they could allow a simplification in the form of the rule, where the histories are not used as input:

$$\Delta\theta_t^{(ij)} = \Phi(o_{t+1}^{(i)}, o_{t+1}^{(j)}, o_t^{(i)}, o_t^{(j)}, \theta_t^{(ij)}, r_{t+1}) \quad (2)$$

A further simplification can be made by not using the "bootstrapping" approach described above, i.e., the pre- and post-synaptic activations from time $t+1$ are not used

as input. This additionally reduces the search space of evolutionary optimization. As in the policy network, the architecture of the learning rule network is restricted to one that does not contain any hidden nodes (in order to have low complexity), but has a recurrent connection on its output node. This is done in order to allow complex dependencies between weight updates. Therefore, the rule used in this study has the following form:

$$\Delta\theta_t^{(ij)} = \Phi(o_t^{(i)}, o_t^{(j)}, \theta_t^{(ij)}, r_{t+1}) \tag{3}$$

At every time step t, the agent chooses an action using its policy ANN. Then the local information from all adaptable synaptic connections k of the policy is given to the learning rule. The learning rule has to process all k connections, before t is incremented, thus, while the policy ANN is activated once every t (stimulus-response time scale), the learning rule ANN is activated k times every t (synaptic-modification time scale). It is worth noting that the total number of connections in the policy could be much larger than k, since the policy could contain connections that are not adaptable. For example, in reservoir computing (Lukoševičius & Jaeger, 2009), where the ANNs contain a very large number of sparsely and randomly connected hidden neurons, the hidden to hidden connections are fixed and only the output connections are adaptable.

If we view the learning rule ANN as a graphical model unfolded in time, we can say that the rule effectively causes an "ordered asynchronous update" on the policy. The update is "ordered", due to the fact that all local variables from the policy are processed by the rule in the same order at each time step t, and "asynchronous", because the recurrent connections in the rule ANN modify the state of rule neurons and thus affect the updates of subsequent policy weights, during the synaptic modification steps.

It is worth noting that the rule could be converted to a batch/offline RL rule by delaying the updates after some iterations or at the end of the episode accordingly. Whilst interesting, such rules fall outside the scope of this study. It is also noteworthy that the behaviour of both the policy and the learning rule ANN is deterministic, since the way they operate does not involve any stochastic elements, i.e., the activation functions are deterministic and the recurrent dynamics is deterministic. Preliminary experiments with a softmax activation function at the output layer of the policy network (and consequently a probabilistic selection of actions), or a Boltzmann machine-type sigmoid function (Ackley et al., 1985) at hidden neurons in either the policy or the rule, did not

reveal any advantages, therefore, they are not used in this study.

3 Experimental Setup

3.1 The Tasks

To evaluate the effectiveness of the learning rules, we use three stationary RL benchmark domains, the mountain car (Moore, 1990; Singh & Sutton, 1996), the acrobot (Spong, 1994; Sutton & Barto, 1998) and the cart pole balancing task (Barto et al., 1983; Gomez et al., 2008), as well as a non-stationary (NS) version of the mountain car task (White, 2006). In order to make the tasks more challenging, the agent is provided only with partial state information, i.e., the velocity information is not given to the agent.

In the mountain car task, a car is situated in a valley and needs to drive up the rightmost hill. However, gravity is stronger than the engine, so the car needs to first move backwards towards the opposite hill and then apply full thrust forward in order to be able to reach the goal. The state is composed of the current position of the car (in $[-1.2, 0.6]$) and its velocity (in $[-0.07, 0.07]$), and the actions are a throttle of $\{-1, 0, 1\}$. The car is initialized with a position equal to $-\frac{\pi}{6}$ (i.e., at the bottom of the valley) and a velocity equal to zero at the beginning of each episode, and an episode ends when the car's position becomes greater than 0.5 or after 1000 time steps. The reward is -1 at each time step and 0 at the goal.

The acrobot is a two-link, underactuated robot, roughly analogous to a gymnast swinging on a high bar (Sutton & Barto, 1998). In this task, the state is composed of four continuous variables: two joint positions (both in $[-\pi, \pi]$) and two joint velocities (in $[-4\pi, 4\pi]$ and $[-9\pi, 9\pi]$ respectively). The actions are a torque of $\{-1, 0, 1\}$ applied at the second joint. At the beginning of each episode, the state is initialized with all variables set to zero. An episode ends when the tip (the "feet") reaches above the first joint by an amount equal to the length of one of the links or after 1000 time steps. The reward is -1 at each time step and 0 at the goal.

The cart-pole balancing task consists of a pole hinged to a cart that moves along a track. The objective is to apply forces to the cart at regular intervals, so as to keep

the pole from falling over for as long as possible. The state is composed of the cart position, the cart velocity, the angle between the pole and its vertical position and the angular velocity of the pole. The reward is +1 at each time step and the episode ends when 100k time steps are reached (which is the goal), or if the pole falls past a given angle or if the cart goes off track. At the beginning of each episode, the cart and the pole are reset to their initial positions.

In the NS mountain car task (White, 2006), the force of gravity changes every 50 episodes and in addition, if the agent tries to finish the task with a velocity greater than 0.005, it is penalized with a reward of -100 and the episode ends. At the beginning of each episode, the car is initialized at the bottom of the valley, as in the stationary version of this problem. The difficulty of the problem is controlled by a parameter, p, which takes values in the range $[0, 1]$. Setting $p = 0$ makes gravity weaker and greatly simplifies the problem, while setting $p = 1$ makes gravity stronger and consequently, the agent faces a more difficult problem. Setting $p = 0.5$ we get the dynamics of the original mountain car problem, without considering the extra penalty of -100 discussed above. In order to be able to draw more meaningful conclusions regarding the behaviour of the agents, the force of gravity does not change randomly, but in a controlled manner. More specifically, parameter p changes according to the schedule $\{0, 0.5, 0, 1, 0.5, 1\}$, i.e., $p = 0$ for episodes $0 - 50$, $p = 0.5$ for episodes $51 - 100$, $p = 0$ for episodes $101 - 150$, $p = 1$ for episodes $151 - 200$ and so on. This schedule is periodically repeated every 300 episodes, which cover every possible pair of these three levels of difficulty, i.e., 'easy' when $p = 0$, 'medium' when $p = 0.5$ and 'hard' when $p = 1$. We need to test pairs of difficulty levels, in order to see how the agent adapts to the changing difficulty. The following ordered set describes these pairs: {'easy-medium', 'medium-easy', 'easy-hard', 'hard-medium', 'medium-hard', 'hard-easy'} (which amount to a total of 300 episodes). By adding 50 episodes at the end (thus a total of 350 episodes), we cover any last adaptation that might be needed for the 'easy' scenario, when changing from 'hard' to 'easy'.

3.2 SARSA(λ) with tile coding

We compare the performance of the evolved rules with the performance of an agent that uses SARSA(λ) (Rummery & Niranjan, 1994) with tile-coding[1] (and accumulating traces), which is provided with either partial or full state information. The comparison with the full state information setting is done in order to have a reference as to what a target performance should be in these tasks. It is worth noting that while it is known that SARSA(λ) needs full observability, we experimentally confirmed that it does not converge to a stable behaviour in the partially observable tasks in which the other rules are evaluated. For this reason, to allow the SARSA agents to have better performance in the partially observable scenarios, we extended their inputs to include the immediately previous observation and action, i.e., a delay window of size 1, as in Gomez et al. (2008). Our preliminary experiments confirmed that a delay window of size 1 was sufficient, while larger windows did not offer any benefits, but only slowed down learning.

SARSA is an on-policy algorithm, meaning that its estimation policy and behavior policy are the same, while an off-policy algorithm can have an estimation policy that is different from its behavior policy. In particular, Q-learning (Watkins & Dayan, 1992) can, for example, behave in a random manner, but it estimates a greedy policy. While Q-learning has a convergence guarantee in the tabular case, it may diverge in the case of linear function approximation. SARSA(λ) has a stability guarantee with linear function approximation and does not diverge (Gordon, 2001). It also has to be noted that SARSA(λ) may diverge when combined with non-linear function approximation (Tsitsiklis & Van Roy, 1997). The reasons above explain our rationale behind the choice of SARSA(λ) coupled with a linear function approximator, rather than a neural network.

3.3 The Evolutionary Algorithms

To optimize the parameters of the learning rule ANNs, we use the Cooperative Synapse Neuroevolution (CoSyNe, Gomez et al., 2008) algorithm. We have also created a memetic algorithm that uses the basic CoSyNe algorithm for global search and (1+1)-

[1]The implementation of SARSA(λ) with tile-coding was ported from Richard Sutton's implementation which can be found in http://webdocs.cs.ualberta.ca/ sutton/MountainCar/MountainCar.html (last accessed: 01 January 2013).

CMA-ES (Covariance Matrix Adaptation Evolution Strategies, Suttorp et al., 2009) for local search, as a means of improving the speed of evolution and the performance of the solutions. It works by simply replacing each fitness evaluation with some iterations of the (1+1)-CMA-ES. To the best of our knowledge, such a combination of the aforementioned algorithms (CMA-CoSyNe) has not been investigated before. In the experiments reported in this paper, for the stationary mountain car, the acrobot and the non-stationary mountain car tasks, the basic CoSyNe algorithm is used (i.e., no local search), while for the pole balancing task, the memetic algorithm (CMA-CoSyNe) is used. This decision was made after observing that the basic CoSyNe algorithm did not consistently manage to create rules that solve the pole balancing task. More specifically, when using CoSyNe, we observed that the task was solved in only 3 out of 10 evolutionary trials, whereas with CMA-CoSyNe the task was solved in all 10 trials. While CMA-CoSyNe may have more computational cost, comparing the two algorithms falls outside the scope of this paper.

3.4 Evaluation Methodology

The evaluation methodology consists of two phases: a training/offline phase and a testing/online phase. The training phase is the evolutionary optimization, where a rule is optimized towards the task/domain using multiple independent evolutionary trials and then selecting the best performing individual over these evolutionary trials. The testing phase comes afterwards, where the best performing individual is evaluated in an RL simulation that runs for more episodes and more independent RL trials than the ones used during the fitness evaluations in the training phase. In addition, the online performance of the best rule in the last generation is compared with the online performance of the best rule in the first generation, as well as the performance of SARSA(λ) with tile coding that uses either full or partial state information.

3.5 Configurations

All evolutionary simulations were run for 10 independent trials with 100 generations per trial. For the mountain car and acrobot tasks, each fitness evaluation consists of running one RL trial for 10 episodes and taking the average return over the last 3 episodes.

This was motivated by the fact that a learning algorithm needs some time to optimize a policy, therefore its performance is assessed at some last episodes where it might have converged to an optimal policy for a certain task.

For the pole balancing task, each fitness evaluation consists of running 5 independent RL trials for 10 episodes and taking the average return over the last 3 episodes from all trials. Preliminary experiments showed that by using only 1 RL trial during the fitness evaluation (as in the mountain car and acrobot tasks), occasionally results in inconsistent behaviour during the online evaluation, i.e., different behaviour between RL trials. We believe that this is due to the only source of randomness in this system, i.e., a tie-breaking mechanism in the selection of actions. Averaging over 5 RL trials alleviated this undesirable effect. For the mountain car and the acrobot tasks, more trials did not offer any benefits and only slowed down the fitness evaluations.

By enforcing each simulation to last only 10 episodes during the fitness evaluation of the rules in the stationary tasks, we effectively constrain evolutionary search into finding sets of weights that highly overfit the task, i.e., rules that quickly modify the policy towards the desired performance. An advantage of such a constraint is that each fitness evaluation takes less time to complete, whereas a disadvantage is that the resulting rules might not generalize as well to new environments.

For the NS mountain car task, each fitness evaluation consists of 5 independent RL trials running for 350 episodes (see Section 3.1 for the rationale of this choice) and taking the average return over all episodes and all trials. As in the pole balancing task, more RL trials during the fitness evaluation showed more stability and consistency in the online phase.

For the CoSyNe and the CMA-CoSyNe algorithms, we used a weight permutation probability of 1.0, as used in the original paper of Gomez et al. (2008). For the acrobot and the mountain car tasks, a population size of 200 was used, while for the NS mountain car task, the population size was set to 100. In all three tasks, we use a mutation probability of 0.3 and a scale parameter for the cauchy distributed noise of 0.3. For the pole balancing task, a population size of 80 was used, a mutation probability of 0.4, a scale parameter for the cauchy distributed noise of 0.4, a number of CMA-ES iterations of 20 and an initial step size for CMA-ES of 1.0.

The configuration of SARSA(λ) with tile-coding is as follows. In all tasks we use

the standard gridlike tilings, as well as hashing to a vector of dimension 10^6. After some experimentation, the number of tilings was set to: (i) 12 for the fully-observable mountain car task, for both the fully- and partially-observable acrobot task, and for the partially-observable pole balancing task, (ii) 20 for the partially-observable mountain car task and both versions of the NS mountain car task, and (iii) 10 for the fully-observable pole balancing task. The number of discretization intervals per observation variable for each tiling was set to 16. Additionally, we use a discount factor $\gamma = 1.0$ in all tasks, since preliminary experiments showed that smaller values resulted in inferior performance. This is not surprising, since all problems considered in this paper are episodic and each observed reward is equally important to express the goal of the respective RL task. For the mountain car and the acrobot tasks, for both the fully- and the partially-observable scenarios, we use a step size $\alpha = 0.2$, eligibility trace decay $\lambda = 0.9$, $\epsilon = 0.0$ for ϵ-greedy exploration, meaning no random exploratory actions. The latter does not mean that there is only exploitation in the system. Exploration comes from the fact that we perform optimistic initialization of the value function, by setting the initial parameters to the maximum possible Q-value of $r_{max}/(1-\gamma) = 0$, where $r_{max} = 0$ is the maximum possible reward in these tasks. We have experimented with larger values, i.e., powers of 10, and up until 10^5 learning was possible; at 10^6 each episode reached the maximum number of steps. For the fully-observable pole balancing task, we use $\alpha = 0.9$, $\lambda = 0.7$, $\epsilon = 0.01$, while for the partially-observable version we use $\alpha = 0.3$, $\lambda = 0.9$, $\epsilon = 0.1$. For the NS mountain car task, we used $\alpha = 0.1$ in the fully-observable version and $\alpha = 0.15$ in the partially-observable version, while $\lambda = 0.95$ and $\epsilon = 0.05$ in both versions. All parameters were chosen after some experimentation and are summarized in Table 3 in Appendix A.

In the partially observable case of SARSA, the observation and action at time $t-1$ are used as part of the features (as discussed in Section 3.2), with the exception of the acrobot task where we have only included the observation and not the action, since this resulted in better performance. The (discrete, 1-dimensional) actions are treated as a continuous variable that has the range $[0, n-1]$, where n is the number of actions in each scenario, and discretized using a number of discretizations equal to n.

During the testing phase, the best performing learning rule ANN of generation 0 and 100, as well as both SARSA versions, are evaluated for 30 RL trials with 1000 episodes

per trial in the stationary tasks and 3000 episodes per trial in the NS mountain car task.

4 Results

Figure 1 illustrates both phases of our evaluation. The left column shows the training/offline phase, i.e., the evolutionary optimization of the rules, in each of the four scenarios, using error bars that represent the standard deviation between the evolutionary trials. Note that in the case of the pole balancing task, the standard deviation is very large in the initial generations and this is due to the fact that some of the trials have already reached the target performance of 100K. At generation 86 the target performance of 100K is reached in all trials, thus, the standard deviation is 0. On the right column, the online performance of the rules is demonstrated for each scenario respectively. We show four types of lines in the graphs of the right column that illustrate the behaviour of the compared RL rules: a) the best performing rule ANN of generation 100 is shown using *black-solid* lines, b) the best performing rule ANN of generation 0 is shown using *grey-solid* lines, c) SARSA with full state information is shown using *black-dotted* lines, and d) SARSA with partial state information is shown using *grey-dotted* lines.

In the mountain car task, we notice that the best performing rule of generation 100 quickly rises from an average return of -179 to -103 from the 2nd episode and maintains that performance until the end of all episodes. Similarly, the best performing rule of generation 0 rises from a value of -244 to -158 from the 2nd episode as well and does not change afterwards. SARSA with full observability converges to an average return of -103.6 at episode 345, while SARSA with partial observability fluctuates at an average value of -143.2 at the last 200 episodes.

In the acrobot task, the evolved rule starts with an average return of -89.98 and converges to a value of -74 from the 2nd episode. In contrast, the best performing rule of generation 0 starts behaving in an oscillating manner from the 3rd episode, by alternating between the values -191.2 and -168.8; this is repeated until the end of all episodes. SARSA with full observability converges to an average value of -67.8 over the last 200 episodes, and displays some minor fluctuations due to averaging between multiple trials. SARSA with partial observability behaves similarly reaching, however, an average value of -77.3 over the last 200 episodes.

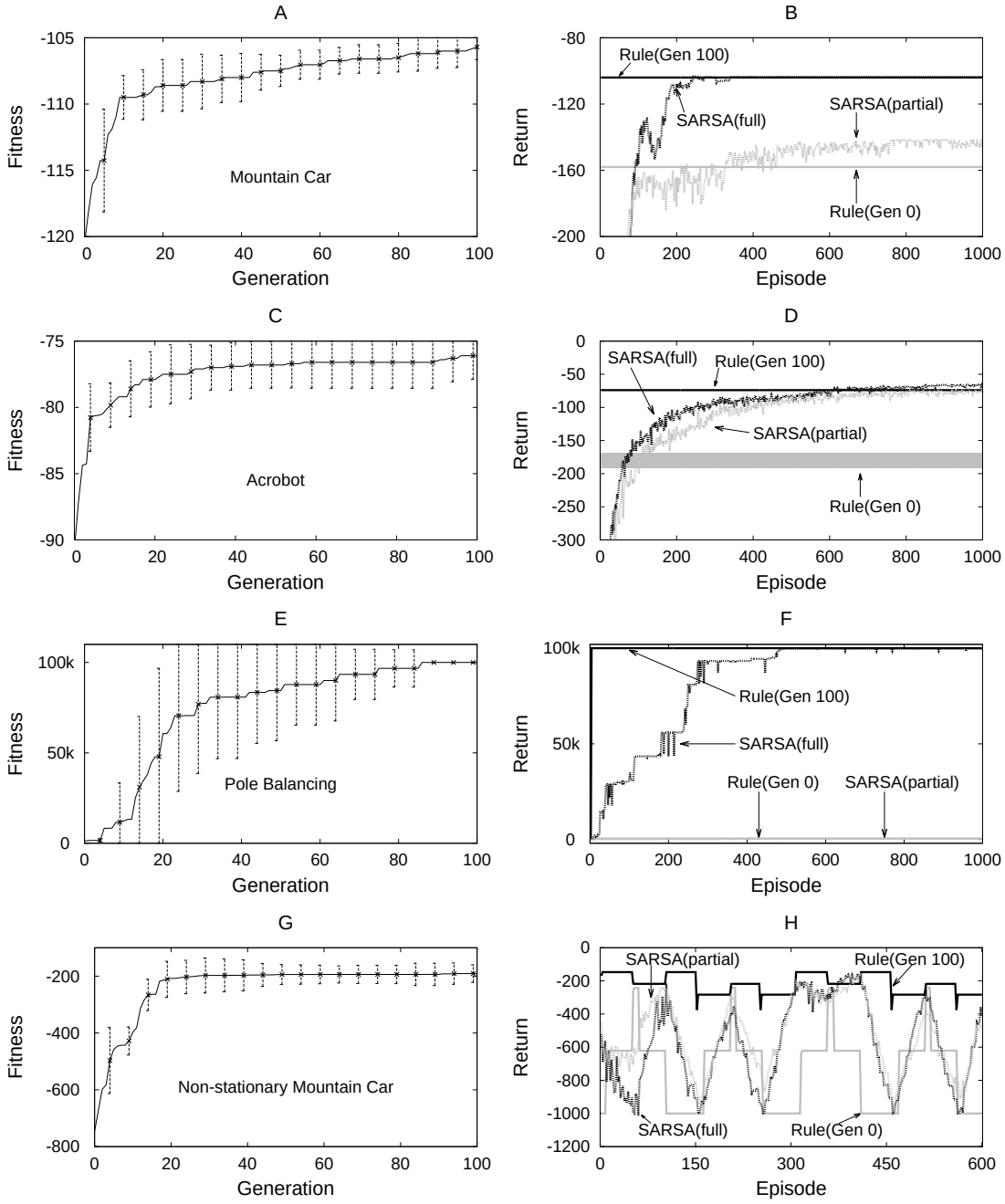

Figure 1: The left column shows the evolutionary results (averaged over 10 independent runs). For each domain of the left column, the online evaluation (averaged over 30 independent runs) is illustrated in the right column, by showing a comparison of the best performing rule at generation 100 (*black-solid* lines), with the best performing rule at generation 0 (*grey-solid* lines), as well as with SARSA that has full observability (*black-dotted* lines) and partial observability (*grey-dotted* lines). The ANN rules receive only partial state information. See text for further details. (A color version of this figure is available in the supplementary material.)

In the cart pole balancing task, the evolved rule converges to the desired performance of 100K from as early as the 4th episode. In contrast SARSA with full observability does reach the desired performance, but not in all RL trials. More specifically, in 27 out of 30 trials SARSA managed to converge, whereas in the remaining 3, it did not. For this reason, the average return of the last 200 episodes is 99.8K, instead of the full 100K. The best performing rule of generation 0, as well as SARSA with partial observability behave very poorly in this task as they accumulate an average return of 457.4 and 415.2 respectively over the last 200 episodes. We have not managed to find good parameter settings that give reliable results for SARSA with partial observability in this task.

The most interesting case is the one of the NS mountain car task (shown at the bottom right part of Figure 1), since the force of gravity changes in a controlled manner every 50 episodes. The reader is reminded that the level of difficulty follows the schedule: 'easy $\rightarrow$ medium $\rightarrow$ easy $\rightarrow$ hard $\rightarrow$ medium $\rightarrow$ hard'. Since each difficulty level lasts 50 episodes, this schedule is repeated every 300 episodes (not to be confused with the 350 episodes we used during the fitness calculation; see Section 3.1 for the rationale behind the additional 50 episodes). The graph shows only the first 600 episodes (with a total of 3000) for convenience. Figure 2A shows a larger version of this graph that contains the first 1500 episodes.

We notice that the best individual of generation 0 starts with a reward of -1000 which means that it has not managed to solve the task, since 1000 is the maximum allowed number of steps (and each step has a reward of -1). After approximately 10 episodes it reaches a performance of -621.3 and remains there until the difficulty level changes to 'medium'. At a medium difficulty level we notice a spike to an average return of approximately -242.5 that lasts for 8 episodes and then a sudden drop to an average return of -621.3 as before and subsequently at -1000. The noticeable trend is that when the difficulty is at a 'medium' level, this rule performs its best, but only for 8 episodes.

The best individual of generation 100 has a radically different behaviour. It starts in the 'easy' difficulty level with an average return of -846 and after 4 episodes converges to a value of -147. As mentioned in Section 3.1, the NS mountain car task has the extra requirement that the agent needs to slow down at the goal (i.e., velocity should be less than 0.005), otherwise it receives a reward of -100. Examining the number of steps

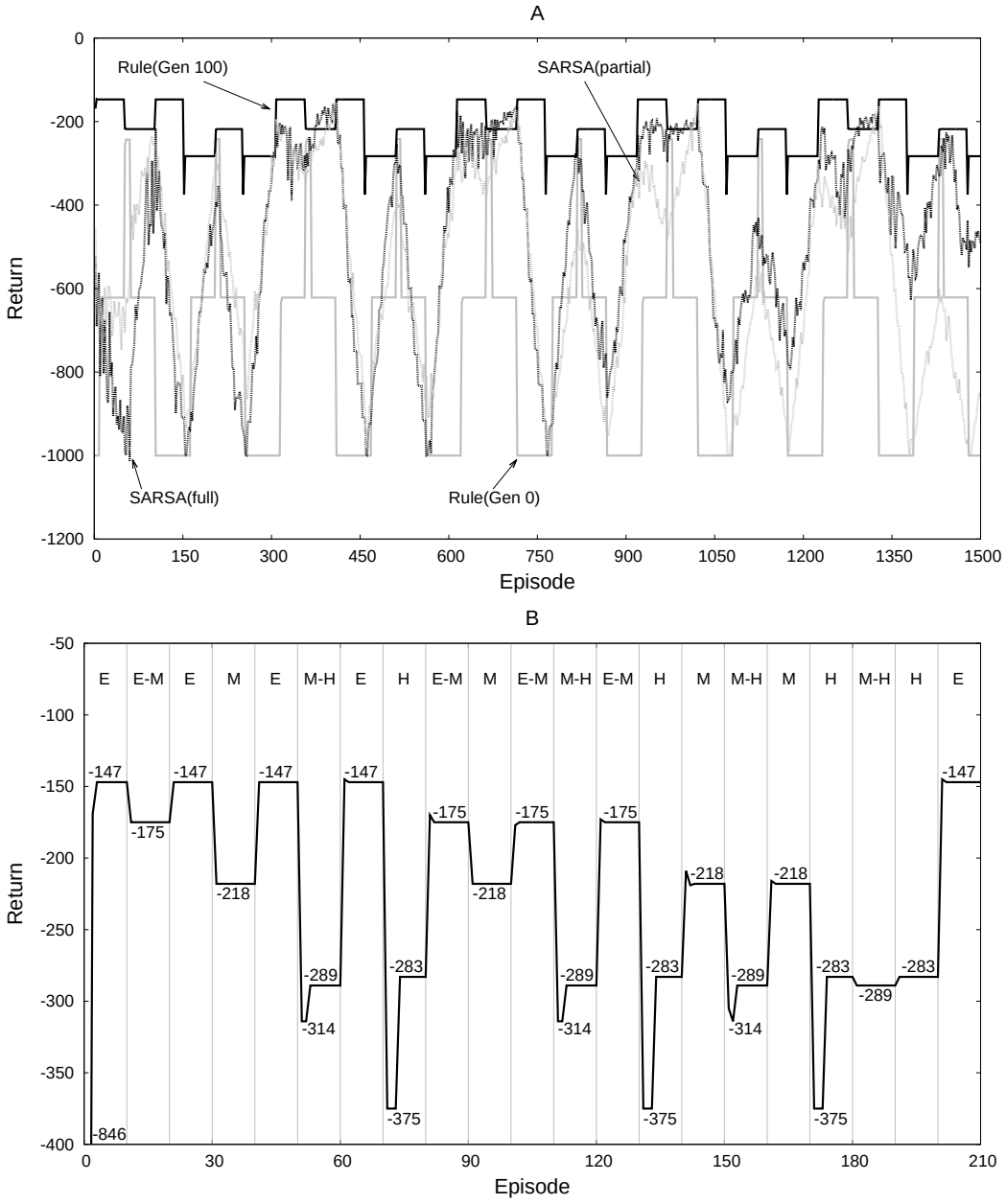

Figure 2: Performance in the non-stationary mountain car task. The performance of all rules for the first 1500 episodes (with a total of 3000) is shown in A. Note that gravity changes every 50 episodes. The (limited) generalization performance of the evolved rule is shown in B where: (i) gravity changes every 10 episodes instead of 50 (that were used during evolution), and (ii) the difficulty level takes not only the 'easy' (E), 'medium' (M) and 'hard' (H) values (that were used during evolution), but the intermediate values 'easy-medium' (E-M) and 'medium-hard' (M-H) as well. (A color version of this figure is available in the supplementary material.)

the agent needs to solve the 'easy' task, we notice that it is 47 instead of 147. This means that the agent managed to solve the task without slowing down at the end and for this reason it received an additional penalty. When the difficulty changes to 'medium', the average return becomes -218 and the number of steps 118. An interesting trend is observed when the difficulty changes to 'hard' at episode 150: a sudden drop to a level of -375 (275 steps) that lasts for 3 episodes and subsequently, a rise to an average return of -283 (183 steps). This is a sign of online adaptation, since the rule seems to create an agent that 'understands' that the difficulty of the task has changed and only needs 3 episodes to adapt its behaviour. Interestingly, we can infer the difficulty level of the task from just observing the performance of the evolved rule: better performance is observed at the 'easy' task, a medium one at the 'medium' task, and its worst performance at the 'hard' task. Another interesting observation is that the evolved rule displays the same behavioural pattern across all 3000 episodes, despite the fact that its fitness evaluation lasted only 350 episodes. While 350 episodes cover every possible pair of difficulty levels (as mentioned in Section 3.1), this does not mean that the evolved rule has optimal performance. This can be observed at some episodes between the episodes 650 and 700, where both SARSA versions manage to receive a larger average return than the evolved rule at the 'medium' difficulty level.

For this reason we performed a set of experiments to identify some empirical bounds that show what the target performance should be for all difficulty levels. This was done by performing experiments with SARSA(λ) with tile coding on two stationary (and fully-observable) versions of the mountain car task. The first is the standard task described in Section 3.1. The second task is similar to the standard one with the added requirement that if the agent finishes the task with a velocity larger than 0.005, it is penalized with a reward of -100. For this reason we name it as "Mountain Car VP" (for Velocity Penalty). It is the same task as the NS mountain car, but without changing the force of gravity throughout the episodes. In order to find the target performance, we varied the parameters of the agent as well as the ones of the simulation (e.g., number of episodes, number of steps per episode etc.) and did not consider averages over RL trials, but the maximum return achieved from all episodes and all trials (without requiring the agent to converge). The results are shown in Table 1. The results suggest that the evolved rule does not manage to achieve the optimal cumulative reward in each

Table 1: Target performance (cumulative reward) for the standard mountain car task and the mountain car task with velocity penalty (VP) for each difficulty level.

Domain	easy	medium	hard
Mountain Car	-44	-101	-141
Mountain Car VP	-62	-111	-159

phase of the problem, since it achieves -147, -218 and -283 for the 'easy', 'medium' and 'hard' phases respectively, as opposed to (the empirically optimal) -62, -111 and -159. However, the number of steps needed to complete the episodes in each phase of the problem are 47, 118 and 183 respectively, since the evolved rule does not manage to create an agent that slows down towards the end of the episode. This means that if the evolved rule is transferred to the standard mountain car task, it will achieve a performance of -47, -118 and -183 in the respective levels of difficulty, as opposed to the (empirically) optimal -44, -101 and -141.

Comparing the behaviour of the two versions of SARSA, we notice that during the first 100 episodes, the one that has partial state information accumulates more reward than the one that is provided with full state information. Whenever the simulation reaches the phase of the 'hard' difficulty level, the performance of both versions of SARSA dips towards a return of -1000, but during that phase it starts rising back up. As time progresses, the performance of SARSA that has full state information does not fall as low as the performance of SARSA that has partial state information, and the former progressively accumulates more reward. Another interesting observation is that when the difficulty level changes from 'hard' to 'medium' or from 'medium' to 'easy', the performance of both agents starts decreasing, whereas, when changing from 'hard' to 'easy' or from 'easy' to 'medium', the performance of both agents continues to increase.

In order to test for some limited generalization in the non-stationary mountain car task, we evaluated the evolved rule in a simulation where: (i) gravity changes every 10 episodes instead of 50 (that were used during evolution), and (ii) the difficulty level takes not only the 'easy' (E, $p = 0$), 'medium' (M, $p = 0.5$) and 'hard' (H, $p = 1$) values (that were used during evolution), but the intermediate values 'easy-medium'

(E-M, $p = 0.25$) and 'medium-hard' (M-H, $p = 0.75$) as well, that were not present during evolutionary training. The performance of the rule is shown in Figure 2B. The simulation lasts 210 episodes, since these are sufficient to cover all 20 possible pairs of the 5 difficulty levels. The first thing we notice is that the performance of the rule in the newly tested E-M and M-H difficulty levels, does not deteriorate. On the contrary, in the E-M level, the performance gracefully interpolates between the respective performance values of the E level (-147) and the M level (-218) to a value of -175. When the phase changes from E to M-H or from E-M to M-H, the performance of the rule falls at -314 for 2 episodes and then converges to -289. In the case where the phase changes from M to M-H the performance falls to -305 in the first episode, then goes to -314 in the second episode and then converges to -289. Although the value -314 is greater than -375 (that corresponds to the level at which the performance drops when the difficulty changes from E to H, from E-M to H, and from M to H), the value -289 (achieved at the M-H level) is less than the value of -283, achieved at the H level. In other words, the performance in the H level is better than the performance in the M-H level. This shows that in the case of M-H, the performance does not interpolate as in the E-M case. We also notice that in the cases where the difficulty changes (i) from M-H to E, (ii) from H to E-M, (iii) from M-H to E-M, (iv) from H to M, (v) from M-H to M, and (vi) from H to E, there is a sudden increase in performance that is slightly better in the first episode (of the respective phase), than in the remaining nine episodes. Lastly, in the cases where the phase changes from H to M-H and from M-H to H, we do not notice any sudden drops or rises in the performance.

Table 2 summarizes the performance of the best rules at generation 0 and 100, as well as SARSA in full- and partial-observability settings, for all tasks. The performance is measured as the average return per episode per trial (over 30 trials) over the last 200 episodes in the stationary tasks (with a total of 1000) and the last 500 episodes in the NS mountain car task (with a total of 3000). Performing a Mann-Whitney U-test (on these last episodes), reveals that the difference in performance between the evolved rules and both versions of SARSA(λ) is statistically significant ($p < 0.001$) in all tasks.

Table 2: Comparison of the performance of the evolved rules and SARSA(λ) with tile-coding. The performance is measured as the average return per episode per trial (over 30 trials) over the last 200 episodes in the stationary tasks (with a total of 1000) and the last 500 episodes in the NS mountain car task (with a total of 3000).

Domain	Rule (Gen 0)	Rule (Gen 100)	SARSA (full)	SARSA (partial)
Mountain Car	-158.0	-104.0	-103.6	-143.2
Acrobot	-179.9	-74.0	-67.8	-77.3
Cart Pole	457.4	100k	99.8k	415.2
NS Mountain Car	-730.2	-219.8	-423.6	-634.6

5 Discussion and Conclusions

In this work, we investigated an approach for representing local learning rules using ANNs. We demonstrated that such a representation can be effective, by finding (through evolutionary optimization) ANN-based rules that perform well in partially observable versions of four environments: the mountain car, the acrobot, the cart pole and the NS mountain car. The evolved rules were compared with the SARSA(λ) learning algorithm which uses a linear function approximator and tile coding for feature extraction. While the results showed that the evolved rules perform better than the version of SARSA that is provided with partial state information, such a comparison is not fair because SARSA is a general algorithm, whereas the evolved rules are very specific to the tasks. Moreover, the evolved rules have 100 generations headstart on the SARSA variants, however, the effect of this overhead would diminish as we increase the number of subsequent tasks being performed, since our goal is to eventually create more general rules. The purpose of the comparison was to show that the ANN representation of the learning rules can be effective in (i) partially observable settings, as its performance was comparable to the performance of a well established learning algorithm which, however, was provided with full state information, and (ii) non-stationary environments, where fast adaptation is needed.

A noticeable trend in all simulations is that the evolved rules converge very fast

and suggest that most optimization occurs offline (i.e., during evolution) rather than online. This is due to the methodology adopted in this study, where the purpose was to overfit the rules to the tasks in order to examine whether such ANN representations are effective. For this reason, if the currently evolved rules are transferred to different domains, they will most probably not perform well. Alternative methodologies, where generalized environments (Whiteson et al., 2011) or multiple domains are used during the fitness evaluation, could possibly show that such ANN-based rules can be used in more general settings. In such methodologies, the number of episodes of the RL simulations (during the fitness calculation) should be large, in order to allow for more online optimization.

During this study, we observed that the problem of designing ANN-based learning rules is hugely multimodal, since there are many different sets of weights that result in the same performance. Moreover, radically different weight vectors produce very similar behaviours, whereas sometimes very similar weight vectors produce disparate behaviours. This issue has not been addressed in this work. Approaches that maintain a behaviourally diverse population (e.g., see Mouret & Doncieux, 2012; Lehman & Stanley, 2011) could explore the search space more efficiently and produce more consistent results.

The general ANN-based learning rules presented in this study can be computationally expensive, because they are queried for every policy ANN connection. Every agent-environment time step (t) has a computational complexity of $O(n + nm)$, due to the costs of activating the ANN controller with n connections and the ANN learning rule with m connections, respectively. Parallelizing the computation of the update for each policy connection could yield some improvements. However, the current form of our rules causes an asynchronous update on the policy, thus, this type of parallelization cannot be realized. This is due to the fact that the rules use ordinary recurrent connections (also known as unit-delay operators, z^{-1}) that delay a signal one internal time step (i.e., at the synaptic modification scale) and thus, the updates must be done in a serial order. Rules that perform synchronous updates could effectively be parallelized. Such rules could be realized by either not using recurrent connections at all, or by using special recurrent connections that delay the signal n internal time steps (i.e., use n-delay operators, z^{-n}). The expressive power of such synchronous ANN-based rules is yet to be

experimentally determined.

The current form of the rules is particularly relevant to immediate reward domains. It is often the case in RL that there is a delayed reward, being given only at the end of the episode. We believe that there exist certain limited delayed-reward tasks in which the proposed approach will work well and this is mainly due to the use of (i) recurrent connections in both the policy and the rule networks, and (ii) evolutionary optimization algorithms, since they are global optimizers. However, in more complex delayed-reward tasks our methodology might struggle since the evolved rules (i) do not seem to explore a lot, because they are deterministic and severely overfit the environment, and (ii) do not utilize a value function, eligibility traces or any other memory mechanism that is known to deal with the temporal credit assignment problem. They only use a recurrent connection that creates dependencies between weight updates.

A disadvantage of the ANN-based rules is the fact that an ANN is like a black-box, where we cannot easily infer what happens internally. For example in this study, although we know the final rule weights (see Table 4 in Appendix B), we cannot easily say what the rule does exactly. An advantage, however, is that a variety of complex learning rules or behaviours could potentially be realized, due to the inherent universal approximation property of ANNs. An interesting observation can be made about this. While an ANN-based learning rule could theoretically approximate behaviourally some other learning rule, it is possible to create learning rule ANNs that are structurally identical to known learning rules, thus having not only approximate, but identical behaviours. For example, a simple TD learning rule, i.e., $\Delta\theta_t^{(ij)} = \alpha * o_t^{(i)} * (r_{t+1} + \gamma * o_{t+1}^{(j)} - o_t^{(j)})$ can be structurally instantiated in a learning rule network that uses Equation 2 (Section 2), where the pre- and post-synaptic activations from time $t + 1$ are used in the input as well. For such a TD learning rule network to be realized, activation functions such as "multiplication" are also needed (these functions are needed for implementing Hebbian rules as well), thus resulting in higher-order networks (Durbin & Rumelhart, 1989), or compositional pattern producing networks (CPPNs, Stanley, 2007) that could mix summing and multiplicative units. This TD learning rule serves only as a proof of concept, by being an example of a rule that can be used for prediction, not for control, when the input of the feedforward neural controller is the complete state space, no hidden units are used and the activation function of the output unit is the linear function. In other

words, this TD learning rule network can be used when the neural controller becomes identical to the tabular case.

It is noteworthy that CPPNs have also been used for representing learning rules using a variant of the HyperNEAT (Hypercube-based NeuroEvolution of Augmenting Topologies) algorithm, called adaptive-HyperNEAT (Risi & Stanley, 2010). This approach requires the policy ANN to reside in a geometric space and makes the update rule dependent on the coordinates of source and destination neurons of a connection, the activation levels and the current weight value. The adaptive-HyperNEAT does not create learning rules in which the synaptic updates are dependent on (or potentially modulated by) the reward signal, since the reward signal is not given as input to the learning rule as in our model. Moreover, the scope of our study is different from the scope of that work in the sense that our purpose is to eventually create more general learning rules instead of overfitting them to the task.

This work could be viewed as a study that optimizes local learning rules. It has to be noted that there are approaches in the literature that optimize the step size parameter of TD learning rules using various strategies (e.g., see Dabney & Barto, 2012 and references therein). TD learning rules such as SARSA(λ), however, are examples of global rules, due to the calculation of the dot product of the parameter vector and the feature vector. Similar approaches cannot be easily applied in our case, since the evolved ANN-based rules do not have any meta-parameters that can be tuned further. In contrast, SARSA(λ) does have tunable meta-parameters (such as the step size, the discount factor etc.), however, SARSA(λ) is a family of learning rules and each combination of its meta-parameters can be seen as instantiating a different rule. This is true for most RL algorithms, i.e., they are families of global RL rules, thus, in our case one could say that the structure of the ANN rule is a distinct family of local RL rules. Different weight combinations instantiate different rules. Finding a general ANN structure that satisfies certain desirable properties (e.g., contraction mapping, that is, a true learning rule, not just an update rule) would be equivalent to finding a family of local learning rules or local RL algorithms, whose meta-parameters are either all or some of the weights. TWEANNs (Stanley & Miikkulainen, 2002) could be used for such an endeavor, however, designing a good methodology does not seem to be a simple task. A related point is the unification of various learning rules into a single equation and some

work exists already in this front (Gorchetchnikov et al., 2011).

Moving towards the goal of evolving more general learning rules, we identify some issues that need to be taken into consideration. The first has to do with stability, which could be viewed as equivalent to generalization (Shalev-Shwartz et al., 2010). Intuitively, stability notions measure the sensitivity of the learning rules to perturbations in the training dataset. The training set in this case contains environments and their parameters. An open research direction therefore for the future could be the investigation of methodologies or experiments in which evolved rules are stable in multiple environments. A second issue is convergence. In TD learning, convergence can be seen in the form of a decreasing TD error, especially in stationary environments, where the error can approach zero. In this work, convergence was not addressed explicitly, even though the fitness function for stationary environments took into account the performance in the last episodes of the simulations. One way to explicitly account for convergence could be to check whether the magnitude of the weight changes produced by the learning rule gets minimized. An approach would be to view convergence as another objective and use a multiobjective optimization algorithm to create rules that are both high performing and convergent. A further issue is optimality. In more general environments, the rules need to be able to address the exploration/exploitation problem online. In this work, this problem was addressed mostly offline, by evolutionary adaptation, and as a result, the rules became very specific to the tasks. How experiments can be designed in which local RL rules emerge that behave optimally in a no-regret, a Bayesian or a Probably Approximately Correct (PAC) sense (Li, 2009), remains an open question. The focus of future work in this area should be to explore such interesting research avenues.

Appendix A: Parameter Settings

Table 3: Parameter settings for SARSA: 'full' denotes the fully-observable and 'partial' denotes the partially-observable version of the task, α is the step-size, γ is the discount factor, λ is the eligibility trace decay rate, ϵ is the probability of selecting a random action instead of the greedy one, nT is the number of tilings, nD is the number of discretizations invervals per observation variable and w is the size of the delay window.

Domain	α	γ	λ	ϵ	nT	nD	w
Mountain Car (full)	0.2	1	0.9	0	12	16	0
Mountain Car (partial)	0.2	1	0.9	0	20	16	1
Acrobot (full)	0.2	1	0.9	0	12	16	0
Acrobot (partial)	0.2	1	0.9	0	12	16	1
Cart Pole (full)	0.9	1	0.7	0.01	10	16	0
Cart Pole (partial)	0.3	1	0.9	0.1	12	16	1
NS Mountain Car (full)	0.1	1	0.95	0.05	20	16	0
NS Mountain Car (partial)	0.15	1	0.95	0.05	20	16	1

Appendix B: Final Evolved Rule Weights

Table 4: Final Evolved Rule Weights (rounded up to 3 decimal points). The arrow ($\rightarrow$) denotes a connection between two neurons in the learning rule network: 'pre' denotes the neuronal input that is responsible for feeding the pre-synaptic activation, 'post' is the neuronal input that feeds the post-synaptic activation, θ is the input that feeds the weight parameter of the currently queried connection in the controller, r is the input responsible for the (normalized) reward the agent obtained at time $t + 1$, 'bias' has a constant value of 1.0, and 'out' is the output neuron.

Source→Destination	Mountain Car	Acrobot	Cart Pole	NS Mountain Car
pre→out	0.397	-0.036	19.139	0.043
post→out	-0.768	0.115	-14.770	0.338
θ→out	-0.423	-0.113	-29.677	-0.451
r→out	-0.083	-0.118	12.448	0.633
bias→out	0.391	2.077	-38.158	0.921
out→out	0.027	0.066	-2.903	-0.191

Acknowledgments

We gratefully acknowledge the support of the University of Cyprus for an internal research project grant. We are also grateful to the three anonymous referees for their constructive and stimulating reviews.

References

Ackley, D., Hinton, G. & Sejnowski, T. (1985). A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, *9*(1), 147–169.

Ay, N., Bernigau, H., Der, R. & Prokopenko, M. (2012). Information-driven self-organization: the dynamical system approach to autonomous robot behavior. *Theory in Biosciences*, *131*(3), 161–179.

Barto, A. G., Sutton, R. S. & Anderson, C. W. (1983). Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE Transcations on Systems, Man and Cybernetics*, *13*(5), 834–846.

Dabney, W. & Barto, A. G. (2012). Adaptive Step-Size for Online Temporal Difference Learning. In: Jörg Hoffmann, Bart Selman (Eds.), *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Durbin, R. & Rumelhart, D. (1989). Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks. *Neural Computation*, *1*(1), 133–142.

Fahlman, S. E. & Lebiere, C. (1990). The cascade-correlation learning architecture. In: D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*, 524–532. San Francisco, CA: Morgan Kaufmann.

Floreano, D., Dürr, P. & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, *1*(1), 47–62.

Funahashi, K. & Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, *6*(6), 801–806.

Gomez, F., Schmidhuber, J. & Miikkulainen, R. (2008). Accelerated Neural Evolution Through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research*, *9*, 937–965.

Gorchetchnikov, A., Versace, M., Ames, H., Chandler, B., Léveillé, J., Livitz, G., Mingolla, E., Snider, G., Amerson, R., Carter, D., Abdalla, H. & Qureshi, M. S. (2011). Review and Unification of Learning Framework in Cog Ex Machina Platform for Memristive Neuromorphic Hardware. In: *Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN 2011)*, 2601–2608. Piscataway, NJ: IEEE.

Gordon, G. J. (2001). Reinforcement Learning with Function Approximation Converges to a Region. In: Todd K. Leen, Thomas G. Dietterich, Volker Tresp (Eds.), *Advances in Neural Information Processing Systems 13*, 1040-1046. Cambridge, MA: MIT Press.

Hornik, K., Stinchcombe, M. B. & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366.

Kolodziejski, C., Porr, B. & Wörgötter, F. (2008). Mathematical properties of neuronal TD-rules and differential Hebbian learning: a comparison. *Biological Cybernetics*, *98*, 259–272.

Lehman, J. & Stanley, K. O. (2011). Abandoning Objectives: Evolution Through the Search for Novelty Alone. *Evolutionary Computation*, *19*(2), 189–223.

Li, L. (2009). A Unifying Framework for Computational Reinforcement Learning Theory. PhD thesis, Rutgers University.

Lukoševičius, M. & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149.

Moore, A. (1990). Efficient Memory-Based Learning for Robot Control. PhD thesis, University of Cambridge.

Mouret, J.-B. & Doncieux, S. (2012). Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. *Evolutionary Computation*, *20*(1), 91–133.

Park, J. & Sandberg, I. W. (1991). Universal approximation using radial-basis-function networks. *Neural Computation*, *3*(2), 246–257.

Rao, R. P. N. & Sejnowski, T. J. (2001). Spike-Timing-Dependent Hebbian Plasticity as Temporal Difference Learning. *Neural Computation*, *13*(10), 2221–2237.

Risi, S., Hughes, C. E. & Stanley, K. O. (2010). Evolving Plastic Neural Networks with Novelty Search. *Adaptive Behavior*, *18*, 470–491.

Risi, S. & Stanley, K. O. (2010). Indirectly encoding neural plasticity as a pattern of local rules. In: *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior: From Animals to Animats (SAB 2010)*, 533–543. New York, NY: Springer.

Rummery, G. A. & Niranjan, M. (1994). On-line Q-learning using connectionist systems. Cambridge University, Technical Report CUED/F-INFENG/TR 166.

Shalev-Shwartz, S., Shamir, O., Srebro, N. & Sridharan, K. (2010). Learnability, Stability and Uniform Convergence. *Journal of Machine Learning Research*, *11*, 2635–2670.

Singh, S. P. & Sutton, R. S. (1996). Reinforcement Learning With Replacing Eligibility Traces. *Machine Learning*, *22*, 123–158.

Spong, M. W. (1994). Swing up control of the Acrobot. In: *Proceedings of the 1994 IEEE Conference on Robotics and Automation*, 2356–2361. San Diego, CA.

Stanley, K. O. & Miikkulainen, R. (2002). Evolving Neural Networks Through Augmenting Topologies. *Evolutionary Computation*, *10*(2), 99–127.

Stanley, K. O. (2007). Compositional Pattern Producing Networks: A Novel Abstraction of Development. *Genetic Programming and Evolvable Machines*, *8*(2), 131–162.

Sutton, R. S. & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Suttorp, T., Hansen, N. & Igel, C. (2009). Efficient covariance matrix update for variable metric evolution strategies. *Machine Learning*, *75*(2), 167–197.

Tsitsiklis, J. N. & Van Roy, B. (1997). An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674-690.

Watkins, C. J. C. H. & Dayan, P. (1992). Q-Learning. *Machine Learning*, *8*, 279–292.

White, A. (2006). *NIPS Workshop: The First Annual Reinforcement Learning Competition*. Retrieved June 28, 2012, from http://rlai.cs.ualberta.ca/RLAI/rlc.html.

Whiteson, S., Tanner, B., Taylor, M. E. & Stone, P. (2011). Protecting Against Evaluation Overfitting in Empirical Reinforcement Learning. In: *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, 120–127. Piscataway, NJ: IEEE.

Wörgötter, F. & Porr, B. (2005). Temporal sequence learning, prediction, and control: a review of different models and their relation to biological mechanisms. *Neural Computation*, *17*, 245–319.

Yao, X. (1999). Evolving Artificial Neural Networks. *Proceedings of the IEEE*, *87*(9), 1423–1447.