Stochastic Synapse Reinforcement Learning (SSRL)

Syed Naveed Hussain Shah†
Robotics, Evolution, Adaptation, and Learning Laboratory
School of Computer Science
Gallogly College of Engineering
University of Oklahoma
Norman, OK 73019–1101
Email: sayyed.naveed@gmail.com

Dean F. Hougen
Robotics, Evolution, Adaptation, and Learning Laboratory
School of Computer Science
Gallogly College of Engineering
University of Oklahoma
Norman, OK 73019–1101
Email: hougen@ou.edu

***Abstract*—Over the past several decades, reinforcement learning has emerged as one of the major paradigms in machine learning because it allows an agent to learn through interaction with its environment, so long as there is some mechanism by which the agent can gain evaluative feedback on the effects of its actions. However, there are still many open questions as to the most appropriate reinforcement learning approach, particularly for difficult problems such as those dealing with delayed reward, unknown reward structures, continuous state and/or action spaces, perceptual aliasing, and/or environmental change. Here we present a new learning algorithm for these types of difficult problems. It combines the eligibility traces approach to reinforcement learning with artificial neural networks for generalization and pushes the idea of stochastic computations down to the level of the synapse. A proof-of-concept experiment in the domain of robotics demonstrates that the approach has promise.**

I. Introduction

This study proposes a novel reinforcement learning (RL) algorithm for artificial neural networks (ANNs) to learn an episodic task in which there is discrete input with perceptual aliasing, continuous output, delayed reward, an unknown reward structure, and environmental change. Because standard RL methods consider discrete state and action spaces, numerous approaches for combining ANNs or other generalization mechanisms with some form of reward-based learning, often for robotic applications, are common in the literature but no single architecture has emerged as the default approach and many open questions remain [1]–[7]. Indeed, questions as fundamental as whether to use model-based or model-free methods are still strongly debated [3], [7]. This paper builds on previous work using stochastic approaches to RL in ANNs for continuous valued functions [8], [9] and gives a proof of concept for extending stochasticity to the level of the synapse.

A. Background

Artificial neural networks are distributed processing systems containing potentially huge numbers of simple processing nodes connected in massively parallel structures [10]. These generally consist of simplified neuron models interconnected using artificial synapses. The strengths of these connections, known as *weights*, can be used to store knowledge [11].

†Dr. Shah is currently a software engineer at Microsoft Corporation.

Computational models of biological nervous systems show that intricate computations can arise from simple neural circuits [12]. This indicates that for at least moderately complex tasks, even simple feed-forward ANNs may suffice (e.g., [13]). *Feed-forward neural networks*, in which data is manipulated as it passes onward through the network from input to output (with no lateral or backward connections), are often used for approximation or generalization.

Learning is a natural trait of many biological organisms and a logical method for setting appropriate weights in ANNs. *Reinforcement learning* (RL) allows an agent to learn a *policy* of appropriate actions for its environment through direct experience, provided there is some mechanism through which the agent can receive evaluative feedback [14]. However, RL faces many challenges in dealing with complex domains such as robotics [15], [16], which often uses high-dimensional, continuous state and action spaces. For example, while policy improvement is guaranteed in discrete problem domains, it is not guaranteed in continuous domains nor with function-approximation-based policy representations [17].

B. Overview

In this research, stochastic RL in an ANN is used to implement the learning dynamics of an agent in a terminal reward, episodic task scenario with continuous output, perceptual aliasing, and a dynamic environment.

An *episodic task* is one in which the environment is reset when some condition is satisfied, such as when some state is reached or after some number of time steps have passed. Each period between resets is known as an *episode* or a *trial*.

A *terminal reward* is a reward that is given only at the end of an episode. This is a particular form of *delayed reward*, in which an agent needs to take a series of actions before it is able to collect the reward. This is in contrast to *immediate reward*, which is received directly after an action is performed.

Compared to immediate reward, delayed reward requires greater insight when designing algorithms [18]. If an agent takes several actions before a reward is obtained, how can it determine which action(s) were responsible for the reward? This is a *temporal credit assignment problem* [19].

One approach is to consider those elements responsible for the actions to be eligible for change by keeping track of

978-1-5386-2726-6/17/$31.00 ©2017 IEEE

participation in the process with *eligibility traces*. A standard *accumulating trace* adds an eligibility value to the previous value, giving a combined recency and frequency heuristic [20].

When extending RL to continuous spaces using an ANN, multiple synapses may be responsible for the reward at a given time step; each synapse should be given credit accordingly. This is a *structural credit assignment problem* [21].

Continuous spaces also require a mechanism for sampling from the infinite possibilities available. This leads to stochastic RL algorithms for learning real valued functions [8], [9].

There are two main functions of the stochastic learning units that produce continuous output. One is to estimate the correct value of the output for a given input. This estimation may be denoted by a mean of a normal distribution of the unit's activation values. The other is to determine the exploration/exploitation behavior the units should exhibit, which may be controlled by a corresponding standard deviation parameter for the unit.

In previous stochastic RL work, standard deviation was based on the known maximum reward in the environment [8]. Here, the reward structure (including the maximum possible reward) is not known by the agent a priori, which adds to problem difficulty because the agent cannot judge the reward it receives on an absolute scale. Instead, our agent must estimate the expected reward based on experience.

Temporal credit assignment is more difficult in continuous spaces as well. In discrete state-action spaces there is typically a constant eligibility value given to a unit whenever it is active. With discrete spaces, a constant value can be added because a discrete action is either taken or not. However, with continuous actions chosen by sampling from a probability distribution, the value added to a unit's eligibility should vary with the current weight sampled from the distribution.

Perceptual aliasing, which occurs when the mapping between external states and internal states is confounded, has long been known to make learning problems more difficult for RL [22] and also interferes with human learning [23].

A *dynamic environment* is one that changes over time, whether subtly or drastically, rarely or continually. Such changes make learning more challenging as an agent needs the additional ability to unlearn what it has previously learned in order to accommodate new information [24].

II. Approach

The Stochastic Synapse Reinforcement Learning (SSRL) algorithm proposed here is inspired by previous algorithms [8], [9] based on the similarity of the problem domains. However, there are several notable differences in the proposed learning model and methodology. These are: (1) stochastic synaptic units, (2) a sliding window to compute expected reward, and (3) standard deviation traces.

A. Artificial Neural Network Controller

We present a class of ANNs suitable for delayed reward problems using simple feed-forward neural networks. In the proposed networks, binary input units are fully and directly connected to real valued output units via synapses. The weights of each synapse are sampled from continuous probability distributions. The output units' activations are computed as the hyperbolic tangent of the sum of the corresponding weighted inputs for all the synaptic units. Such an ANN is shown in Figure 1, where I represents the number of binary inputs, O represents the number of outputs with real values ranging from -1 to 1, w represents a sampled weight from the weight distribution for each synapse, μ and σ represent the mean and standard deviation (respectively) of the weight distribution for each synapse, and a represents the activation value for each output unit.

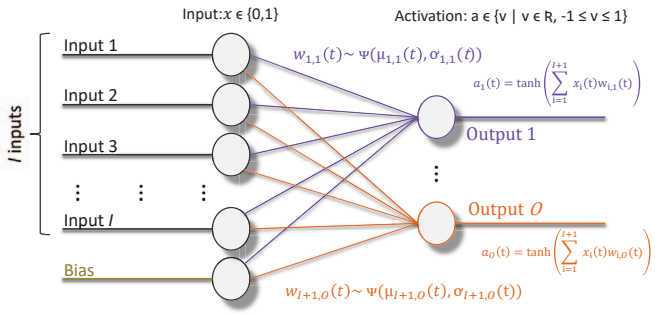


Fig. 1. A fully connected feed-forward neural network with I input units, a bias unit, no hidden units, and O output units.

B. Learning Algorithm Concept

There are two important functions of stochastic learning units producing real valued outputs: (1) determining the mean value to output and (2) determining the degree to which output values should vary [8], [9]. Previous approaches sample from a continuous probability distribution, such as a normal distribution, on a per-unit basis [8], [9]. In contrast, the algorithm proposed in this research samples on a per-synapse basis, which gives a much finer grain to the learning.

The mean of each weight distribution corresponds to a noisy partial policy. When a certain presynaptic unit has a value of one, the synapse contributes to the activation of the postsynaptic unit a value that is likely to be close to the mean. Using a weight-mean update rule, the algorithm estimates appropriate synaptic weight means for the next episode and this probabilistically leads to appropriate mean output values over the course of repeated episodes. Similarly, a synaptic weight-standard-deviation update rule controls the exploration/exploitation trade-off. Each synaptic weight mean μ is updated using Equation 3 and each synaptic weight standard deviation σ using Equation 5.

C. Algorithm Steps

The algorithm is designed for a terminal reward, episodic situation. Generally, there are many time steps in each of many trials during the lifetime of a learning agent. At each time step, SSRL carries out the following steps.[1]

[1]Pseudocode for the complete algorithm is also available [25].

1) Output is calculated based on input by sampling each weight from its normal distribution using

$$w_{ij}(t) \sim \Psi(\mu_{ij}(t), \sigma_{ij}(t)), \tag{1}$$

where $w_{ij}(t)$ is the sampled weight value, $\mu_{ij}(t)$ is the mean of the weight's distribution, and $\sigma_{ij}(t)$ is its standard deviation, for the synapse between input neuron i and output neuron j at time step t. The randomly sampled weight determines whether the policy for the current trial is more exploratory or exploitatory.

2) The activation for each output unit is computed using

$$a_j(t) = \tanh\left(\sum_{i=1}^{I} x_i(t) w_{ij}(t)\right), \tag{2}$$

where $a_j(t)$ is the activation value of output unit j at time step t, $x_i(t)$ is the input to neuron i at time step t, and I is the number of input units in the network.

3) Eligibility increments for these synapses are calculated based on the samples according to Equations 4 and 7.

At the end of an episode, the means and standard deviations of all synapses are updated using Equations 3 and 5 which consider the eligibilities accumulated during the episode.

Expected reward allows the agent to evaluate its performance. The use of a sliding window for recent rewards makes sense as changing policies based on too little experience (for example, by just looking at the last reward collected) causes agents to make inappropriate reward estimates (unless future rewards are based entirely on recent rewards) and thus they do not perform well. Similarly, at the other extreme, paying attention to all previous rewards collected causes the agent to change its policies based on information that is likely to be too old if environment change is expected during the lifetime of the agent. Note that the concept of a sliding window for determining expected reward is a novel contribution compared to previous work [8], [9], which leaves determination of expected reward as an open question.[2]

D. Learning through Weight Adjustments

Using the reward value received at the end of each episode, the algorithm updates the weight-mean parameter for the synapse between input neuron i and output neuron j using

$$\mu_{ij}(\tau+1) = \mu_{ij}(\tau) + \eta_\mu \left(r(\tau) - \overline{r}(\tau)\right) \sum_{k=1}^{t} e_{\mu,ij}(k) d_\mu^{(t-k)}, \tag{3}$$

where $\mu_{ij}(\tau)$ is the synaptic weight mean for trial τ, η_μ is the learning rate, $r(\tau)$ is the reward/penalty collected at the end of trial τ, $\overline{r}(\tau)$ is the average (expected) reward received so far (until trial τ) calculated using a sliding window, and $\sum_{k=1}^{t} e_{\mu,ij}(k) d_\mu^{(t-k)}$ is the sum of all the discounted eligibilities for a particular synaptic weight mean in this trial (symbolized as $\hat{E}_\mu$), and t denotes the time step in a given trial τ. Note that the synaptic weight eligibility values reset at the beginning of each trial. These eligibilities will be referred in this study as *mean eligibility traces*.

Within the summation, $e_{\mu,ij}(k)$ represents the eligibility of a given synaptic weight mean at time step k, d_μ is the discount rate (a constant), and t is the time step at which eligibility is being calculated (the time step at which the trial ends).

The eligibility of a synapse at a given time step k depends on the binary input value of the presynaptic unit and the difference between the sampled weight value on that time step and the mean of the synapse's weight distribution, as follows

$$e_{\mu,ij}(k) = x_i(k)(w_{ij}(k) - \mu_{ij}). \tag{4}$$

Thus the weight adjustment rule can be summarized as follows (also shown in Table I):

If the agent is performing better than expected, the algorithm shifts the synaptic weight means of the corresponding weight distributions in the direction of the discounted sampled weight values that resulted in better performance. On the other hand, if it performed worse than expected, then the algorithm shifts the means in the direction opposite of the discounted sampled weight values.

TABLE I
SUMMARY OF THE LEARNING ALGORITHM—MEAN ADJUSTMENT.

Reward	Eligibility	Mean Adjustment
$r - \overline{r} > 0$	$\hat{E}_\mu > 0$	Increase Mean
$r - \overline{r} > 0$	$\hat{E}_\mu < 0$	Reduce Mean
$r - \overline{r} < 0$	$\hat{E}_\mu > 0$	Reduce Mean
$r - \overline{r} < 0$	$\hat{E}_\mu < 0$	Increase Mean

E. Exploration/Exploitation Trade-off

Besides updating the means of the weight distributions for synapses of the neural network, the algorithm also updates the standard deviation values for the weight distributions for all of the connections between input neurons and output neurons that were active during the current trial. Updating the standard deviations is an important part of the proposed learning algorithm as updating these values effectively determines whether to explore or exploit more during the next trial. This is in contrast to prior work [8] which uses expected reward to compute both the mean and the standard deviation for each neural unit as a whole, rather than calculating a mean and a standard deviation value for each synapse. The equation to update the standard deviation values is

$$\sigma_{ij}(\tau+1) = \begin{cases} 0.05, & \text{if } \sigma_{ij}(\tau) + \Delta\sigma_{ij}(\tau) \leq 0.05 \\ 1, & \text{if } \sigma_{ij}(\tau) + \Delta\sigma_{ij}(\tau) \geq 1 \\ \sigma_{ij}(\tau) + \Delta\sigma_{ij}(\tau), & \text{otherwise,} \end{cases} \tag{5}$$

[2]The sliding window is a queue, which uses more memory than a single expected reward value. However, the benefit to cost ratio is high. Comparative results for various approaches such as average over all previous trials, last reward, and discounted reward will be considered in future work.

TABLE II
SUMMARY OF THE LEARNING ALGORITHM—STANDARD DEVIATION ADJUSTMENT.

Reward	Eligibility	Eligibility Basis	Standard Deviation Adjustment	Resulting Change
$r-\overline{r}>0$	$\hat{E}_\sigma>0$	Exploration	Increase Std. Dev.	More Exploration
$r-\overline{r}>0$	$\hat{E}_\sigma<0$	Exploitation	Reduce Std. Dev.	More Exploitation
$r-\overline{r}<0$	$\hat{E}_\sigma>0$	Exploration	Reduce Std. Dev.	More Exploitation
$r-\overline{r}<0$	$\hat{E}_\sigma<0$	Exploitation	Increase Std. Dev.	More Exploration

where $\sigma_{ij}(\tau)$ is the value of the synaptic weight standard deviation at trial τ, and $\Delta\sigma_{ij}(\tau)$ is the change in value of the synaptic weight standard deviation for a particular synapse.

Equation 5 essentially adds $\Delta\sigma_{ij}(\tau)$ to $\sigma_{ij}(\tau)$ but ensures that $\sigma_{ij}(\tau+1)$ has a lower bound of 0.05 and an upper bound of 1. This helps control the amount of exploration. Even if the algorithm is very successful when exploiting some parts of the environment, it should still explore a little in case there are changes in other parts of the environment. Similarly, there should be an upper limit to the amount of exploration the algorithm permits. If exploration is not capped at the upper end, it becomes difficult for the algorithm to converge back to a reasonable exploration rate even if the sampling becomes conservative.[3]

The change in the standard deviation of the weight distribution is calculated using

$$\Delta\sigma_{ij}(\tau)=\eta_\sigma\left(r(\tau)-\overline{r}(\tau)\right)\sum_{k=1}^{t}e_{\sigma,ij}(k)d_\sigma^{(t-k)}, \qquad (6)$$

where η_σ is the learning rate and $\sum_{k=1}^{t}e_{\sigma,ij}(k)d_\sigma^{(t-k)}$ is the sum of all the discounted eligibilities for the standard deviation from time step 1 to time step t for a particular synapse in this trial (symbolized as $\hat{E}_\sigma$). These eligibilities will be referred to in this research as *standard-deviation eligibility traces*. Note that the standard-deviation eligibility values also reset to 0 at the beginning of each trial.

Within the summation, $e_{\sigma,ij}(k)$ represents the exploration eligibilities of a given synapse at various time steps denoted by k, while d_σ is the discount rate and is constant[4].

As with the eligibility traces for the means of the weight distributions, those synapses that were active more frequently and more recently in the current trial are more eligible for adjustments to their synaptic weight standard deviations which is again different from the prior approach [8] where standard deviation is a monotonically decreasing non-negative function of the expected reward.

The eligibility for change to the standard deviation of a synapse's weight distribution is calculated using

$$e_{\sigma,ij}(k)=x_i(k)(|w_{ij}(k)-\mu_{ij}|\ -\sigma_{ij}), \qquad (7)$$

[3]The lower and upper bound values were arrived at analytically and confirmed experimentally but no rigorous sensitivity testing was performed. Thus the values are not arbitrary but may not be optimal.

[4]Note that this discount rate value is independent of the one used for weight mean adjustment.

Thus the exploration/exploitation rule can be summarized as follows (also shown in Table II):

If the agent is performing better than expected, the algorithm encourages exploration or exploitation, whichever was being used. However, if it performs worse than expected, then the algorithm encourages the opposite of what it had been doing. Contrary to prior work [8], [9], the algorithm controls which policy to follow and how exploratory it should be at the synapse level.

III. EXPERIMENT

We performed a simulation experiment with 30 repetitions to validate SSRL on a moderately complex robotic task that is episodic with terminal reward, perceptual aliasing, continuous output, an unknown reward structure, and environmental change. The robot's task is to explore an arena over a lifetime of 4000 trials of up to 1000 time steps each during which it should learn in order to maximize its lifetime total reward.

A. Experimental Setup

The setup combines elements of the light switching arena [26] and bee foraging field [27] and extends them in a novel setup.

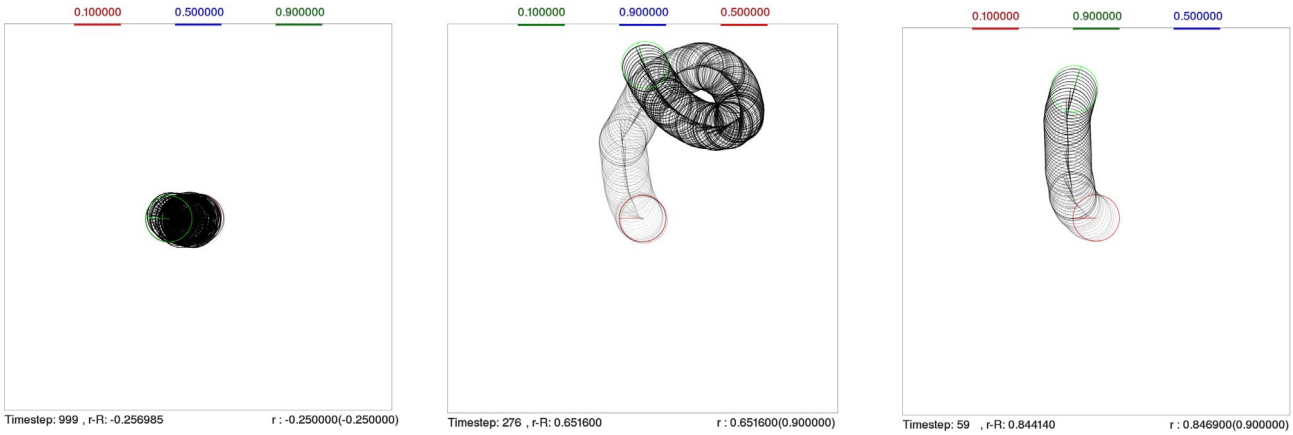
In a square arena, there are three lights of different colors—red, green, and blue—with different reward values—high (0.9), medium (0.5), and low (0.1).[5] At the start of a robot's life, each light color is randomly assigned to one of three positions along one wall, as shown in Figure 2. Once the lights are positioned, the reward values are assigned to them randomly.

The robot, a simulated e-puck robot with two differential wheels and a front-facing linear color camera 60 pixels wide, starts each trial from the center of the arena facing a neutral wall and aims to find a (near) optimal path to the best rewarding light present. If the robot arrives at a light before the trial time expires, the robot collects a reward and the trial ends. The reward value is calculated using

$$r=\frac{t_m-t_c}{t_m}r_v, \qquad (8)$$

where r is the scaled reward received, t_m is the max time step (1000), t_c is the time step on which individual reaches the light, and r_v is the raw reward value of the light (i.e., 0.9, 0.5, or 0.1). If the robot does not reach any light before the trial ends, it gets a penalty of -0.25.

[5]The reward values were chosen to be positive, intuitive, and congruent with other experimental values. For other positive reward values with the same proportional differences, we expect similar results and performance.



(a) Poor performance. The robot spun in circles for the entire trial.

(b) Intermediate performance. The robot reached the high rewarding light but the path contained an extraneous loop.

(c) Near-optimal performance. The robot moved to the high rewarding light at high speed and via a direct path.

Fig. 2. Plots showing performance of the robot. The gray outline shows the walls of the arena and the bars along one wall colored red, green, and blue show the locations of the lights in the arena. Note that the colors and reward values of the lights change between and during lifetimes. Circles show the robot's position while lines show its heading on each time step. The robot's initial position is represented by a red circle and its final position is represented by a green circle. The steps in between are represented by lighter gray (earlier steps) to darker gray (later steps). Below each arena figure, "Timestep" shows the total number of time steps taken by the robot during that trial, "r-R" shows the current scaled reward minus the average reward and "r" shows the current scaled reward received and, parenthetically, the raw reward for the light reached. In (a), no light was reached so the reward value was -0.25, whereas for (b) and (c) the high rewarding light was reached in each case.

After a trial ends, the arena is reset and the next trial begins. Half way through the robot's lifetime, the high reward value is swapped with the low reward value to change the environment. It is important to note that the best an agent with no learning capability could do would be to visit the same light over and over again or to approximate this by selecting a random light each time. Examples of robot performance in the arena are shown in Figure 2.

B. Artificial Neural Network Controller

The robot is controlled by an ANN that belongs to the same class shown in Figure 1. The controller has 19 binary inputs, 18 representing camera data plus 1 bias. The inputs are fully connected to 2 continuous output units, 1 to determine the speed of each wheel.

The camera data indicates the presence or absence of colors in different camera regions. The camera's visual field is divided into 6 regions: far left, left, near left, near right, right, and far right. If 5 or more (out of 10) of the pixels in a given region are a given color then an input of 1 is given to the corresponding neuron. Otherwise, an input of 0 is passed to that neuron. (6 regions times 3 colors gives 18 sensory inputs.)

This sensing setup is robot-centric rather than world-centric. The robot does not know its own x, y, or θ world coordinates nor the locations of the lights or walls in world coordinates. Rather, it knows what it can sense. For example, it might sense the green light in its right visual region. The robot might be close to the green light or far away and still receive that same visual input. This results in perceptual aliasing.

The robot is determined to have reached the light (thus ending the trial), when all 6 of its visual input regions register the presence of the same light.

C. Learning

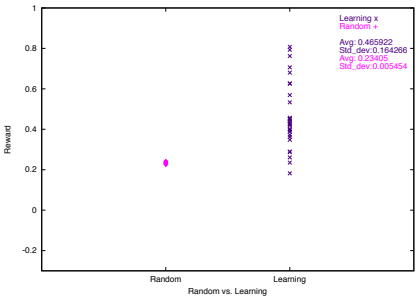
A good learning agent will look for the best rewarding light and, once discovered, will exploit that resource until the environment changes. The agent will then explore again to find the new best rewarding light. Once found, it will again exploit that resource.

At the beginning of each lifetime, all ANN synaptic weight means are initialized randomly between 0 and 1. The range -1 to 0 is avoided initially as they correspond to the robot moving backward, which is generally ineffective since the camera points forward. These means are denoted μ. Note that in the random case (the control condition used for comparison), the synaptic weight means are randomly initialized at the beginning of every trial.

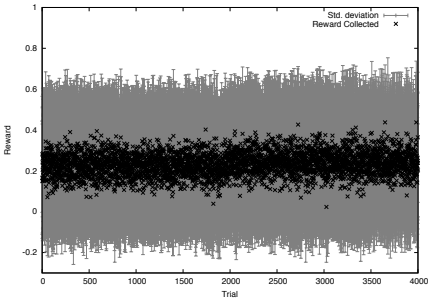
At the beginning of each lifetime all ANN synaptic weight standard deviations, denoted σ, are initialized to a constant value of 0.9. It is important to note that the robot should start aggressively exploring the arena, thus a high initial value for each σ makes sense. Further, again for the random case, the synaptic weight standard deviations are randomly initialized at the beginning of every single trial.

The real valued activations $[-1,1]$ are linearly scaled to $[-15,15]$, the range of values between full reverse and full forward for e-puck robots.

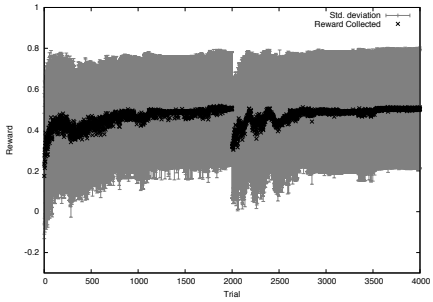
A sliding window of 20% of the total number of trials is used to compute the average of the most recent rewards collected. At the beginning of the lifetime of the individual, all the entries in the window are initialized to 0. This sliding window acts as a queue (FIFO). After each trial is over, the scaled reward collected is inserted while the oldest drops out. Thus, over several trials, this queue builds up recent rewards.



(a) Average rewards collected for 30 repetitions of 4000 trials each (random ANN vs. learning).



(b) Random Condition (Average). Mean and standard deviation of the reward collected by 30 agents across a lifetime (4000 trials).



(c) Learning Condition (Average). Mean and standard deviation of the reward collected by 30 agents across a lifetime (4000 trials).

Fig. 3. Average performance.

The average reward is always computed over all the values in the queue. Thus, during the beginning phase of the individual's lifetime, it's expectation from the surrounding world is very low. As the number of trials progresses, the robot's reward expectation depends more on its past experiences. The main benefit of using this window is to ensure that frequent bad experiences in the beginning about the surrounding world should not unduly influence the robot's understanding of the world as its expectation initially will always be better than failure (-0.25 vs. ≈ 0). Similarly, if it happens to collect a positive reward (any of the three rewards) it is encouraged by positive experiences, since these are greater than 0. This window gives a reasonably large opportunity for the simulated robot to learn about the environment before it decides to exploit a particular policy. It is important to note that the only reward information with which the agent begins is that positive values are reward, negative values are punishment, and 0 is neutral.

The other SSRL parameters used in this experiment (η_μ, η_σ, d_μ, and d_σ) are all set to 0.5.

IV. Results

Figure 3a summarizes the results of learning versus random behavior for the given task. It shows that the learning algorithm outperformed random neural means and standard deviations on 28 out of 30 repetitions and that the average reward earned for learning (0.465) is more than twice that for random (0.23). These results are statistically significant (t-test, $p < 0.0001$).

Looking at the results for random agents, Figure 3b shows the average reward collected across 30 repetitions of each trial. As expected, the graph demonstrates poor average performance without discernible improvement.

Figure 4a shows typical results for a random agent in the arena. It almost equally tries all lights throughout its lifetime regardless of reward received.[6] Likewise, it frequently fails to reach any light before a trial ends, resulting in the agent receiving many penalties throughout its lifetime. As the weight means and standard deviations are initialized to random values every trial, these behaviors are expected.

Moving on to results for learning agents, Figure 3c shows the average reward received across 30 repetitions of each trial. The graph shows that there is an upward trend of learning in both halves of the lifetime of the agents. A drop in average reward collected can be noticed at trial 2000, due to the switch in the high- and low-rewarding lights, as expected.

Unlike the random agents that all behave very similarly to one another, there are a variety of behaviors exhibited by the agents that learn.

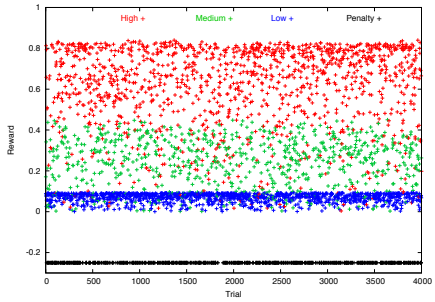
Figure 4b presents a typical good learner that explores the arena a little in the beginning of its lifetime and then quickly focuses on the best-rewarding light. Half way through the lifetime, when the reward for the light being exploited is switched from high to low, it quickly adjusts to the new high-rewarding light and updates its policy to get there quickly.

Figure 4c depicts a typical moderate-learning case where the agent has good initial random weights for going to the medium-rewarding light yet quickly finds the high-rewarding light and learns a policy to exploit that. However, once the change in reward happens at 2000 trials, the agent fails for a few trials before learning to get to a better light. In this case, the agent never exploits the high-rewarding light during the second half of its lifetime even though it does encounter that light during that period; however, the agent finds the medium-rewarding light and exploits that.

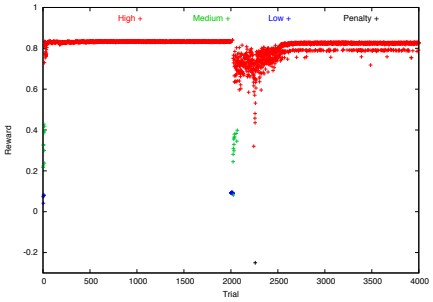
Figure 4d shows another typical moderate-learning case but of a different type. This agent explores all three lights in the beginning of its lifetime and then chooses the medium-rewarding light. Since the medium-rewarding light offers a consistent reward throughout the lifetime of the agent, the agent performs moderately throughout its lifetime and does not alter its behavior when the low and high-rewarding lights swap values at Trial 2000.

Figure 4e shows a poor-learning case where the agent experiences a low-rewarding light as well as failures and learns to go to the low-rewarding light, thus at least learning to avoid the penalty. However, it never explores sufficiently to discover

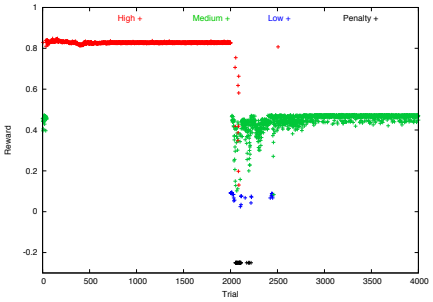
[6] In all agent performance graphs, red represents reward from the high-rewarding light (base=0.9), green represents reward from the medium-rewarding light (base=0.5), and blue represents reward from the low-rewarding light (base=0.1), not the actual color of the lights in the arena.



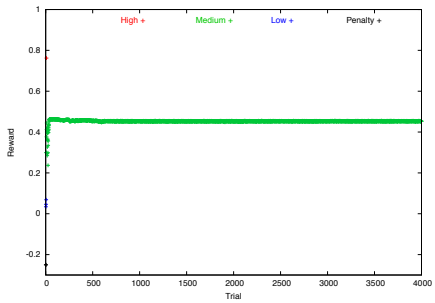
(a) Random Condition (Typical). Reward collected in each of 4000 trials by an agent with random weights.



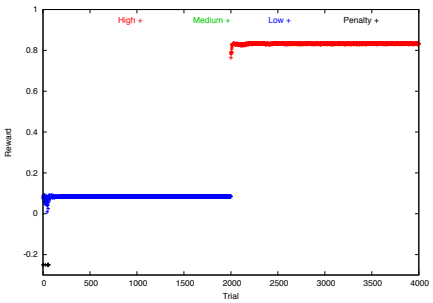
(b) Learning Condition (Typical Good Learner). Reward collected in each of 4000 trials.



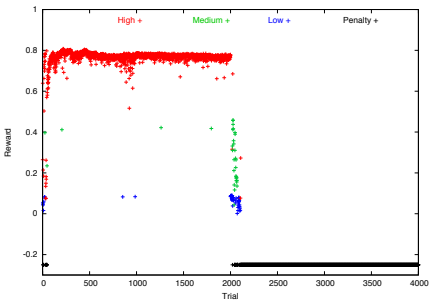
(c) Learning Condition (Typical Moderate Learner). Reward collected in each of 4000 trials.



(d) Learning Condition (Typical Moderate Learner). Reward collected in each of 4000 trials.



(e) Learning Condition (Typical Poor Learner). Reward collected in each of 4000 trials.



(f) Learning Condition (Typical Failed Learner). Reward collected in each of 4000 trials.

Fig. 4. Plots of typical performance. The criteria for good, moderate, and poor are based on an ideal non-learner's average reward during its lifetime. Good means outperforming this agent in both halves of its lifetime, moderate means exactly one half and overall, and poor means not outperforming in either.

the medium- or high-rewarding lights and continues to exploit the low reward until the change in the environment changes the low-rewarding light to high-rewarding.

Finally, Figure 4f shows one of the two agents that did not exhibit much learning. Here, early exploration of all the lights and exploitation of the high-rewarding light in the first half of its life makes this agent a good learner during that half of its life. However, this agent is not able to cope with the change and its weight adjustments result in consistently poor behavior very soon thereafter.

V. Discussion and Conclusions

This work contributes to robotics, machine learning, and potentially computational neuroscience. It introduces SSRL, which works well for an episodic task in a changing environment with perceptual aliasing, continuous output, terminal reward, and an unknown reward structure. The algorithm demonstrates good performance using synaptic weight standard-deviation eligibility traces which take into account past actions to refine the exploration/exploitation strategy.

This study shows that using SSRL, a simple fully connected feed-forward neural network with no hidden units is powerful enough to find near-optimal paths in this space despite the many complicating factors of the scenario. Of course, performance cannot be guaranteed under these conditions [17].

SSRL uses the novel idea of a sliding window to help the agent use its prior knowledge in an effective way to decide future policies. This algorithm uses structural as well as temporal credit-assignment-based stochastic-synaptic-weight mean adjustment and stochastic-synaptic-weight standard-deviation adjustment as two major novel approaches. The synaptic weight standard-deviation adjustment acts as a control knob for the exploration/exploitation trade-off at the synapse level. Controlling the knob by considering past actions in the state-action space, an algorithm capable of learning the aforementioned task with near-optimal solutions is devised, although at a greater cost that using stochasticity at the neuron level. The statistically significant results show that SSRL has potential.

VI. Future Work

The approach of stochastic synaptic weights introduced here contrasts with that of the stochastic activation units approach [8], [9]. In the future, comparisons can be made between the approaches to determine if one outperforms the other in particular domains. In addition, while SSRL was designed for and tested here with discrete inputs, we see no fundamental reason it should not also work well for continuous inputs, so that should be tested as well.

To understand and confirm the generalizability, scalability, and robustness of SSRL, an agent can be given a different task to learn. An example of such a task can be a parent learning to become a nurturer for its offspring.[7] This would

[7]This choice is made to contribute to a larger research agenda involving the evolution or nurturing and learning [28].

mean, for example, that an arena could be designed in which there are several light switches on one end that activate a single light source on the other end. The reward value of the light would be determined by which switch is turned on by the nurturer. The parent's (nurturer's) job would be to choose the right switch to turn on in order to provide maximum reward to its offspring. That would also mean that a communication mechanism between the child and the parent must exist and, preferably, be evolved. Based on the feedback from the child, the parent should improve on its behavior and make intelligent decisions over its lifetime. A reward switch between and during each lifetimes would make this a dynamic environment.

Further, consider an experimental setup containing resources with high risk, high reward; moderate risk, moderate reward; and low risk, low reward. This could give three different options that have the same mean reward but different levels of risk (in the sense of economics [29]). Because risk aversion is a side effect of reinforcement learning [27], individuals should learn to visit the (currently) least risky resource most often. Comparisons of our proposed algorithm with state of the art reinforcement learning algorithms as well as to similar algorithms [8], [9] could yield useful insights. Further, variations of this algorithm can be parameterized and evolutionary algorithms used to find maxima in a solution space using a carefully designed fitness-based genetic algorithm [30].

ACKNOWLEDGMENTS

The authors gratefully acknowledge substantial supercomputing resources, including outstanding personal assistance, provided by the OU Supercomputing Center for Education & Research (OSCER) and the Department of Computer Science, without which this research would not have been possible.

REFERENCES

[1] H. van Hasselt, "Reinforcement learning in continuous state and action spaces," in *Reinforcement Learning: State-of-the-Art*, M. Wiering and M. van Otterlo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 207–251. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_7

[2] A. Ghanbari, Y. Vaghei, S. Noorani, and S. M. Reza, "Reinforcement learning in neural networks: A survey," *International Journal of Advanced Biological and Biomedical Research*, vol. 2, no. 5, pp. 1398–1416, 2014. [Online]. Available: http://www.ijabbr.com/article_7340.html

[3] J. Kober and J. Peters, "Reinforcement learning in robotics: A survey," in *Learning Motor Skills*. Cham: Springer International Publishing, 2014, vol. 97, pp. 9–67. [Online]. Available: http://link.springer.com/10.1007/978-3-319-03194-1_2

[4] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0893608014002135

[5] S. Amarjyoti, "Deep reinforcement learning for robotic manipulation: The state of the art," *arXiv preprint arXiv:1701.08878*, 2017. [Online]. Available: https://arxiv.org/abs/1701.08878

[6] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274v3*, p. 66 pages, Jul. 2017.

[7] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, May 2017. [Online]. Available: http://link.springer.com/10.1007/s10846-017-0468-y

[8] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Networks*, vol. 3, no. 6, pp. 671–692, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809090056Q

[9] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992. [Online]. Available: http://link.springer.com/article/10.1007/BF00992696

[10] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[11] I. W. Sandberg, J. T. Lo, C. L. Fancourt, J. C. Principe, S. Katagiri, and S. S. Haykin, *Nonlinear dynamical systems: Feedforward neural network perspectives*, ser. Adaptive and Learning Systems for Signal Processing, Communications and Control. John Wiley & Sons, 2001, vol. 21.

[12] J.-M. Fellous and C. Linster, "Computational models of neuromodulation," *Neural Computation*, vol. 10, no. 4, pp. 771–805, May 1998. [Online]. Available: http://dx.doi.org/10.1162/089976698300017476

[13] A. Leonce, B. Hoke, and D. Hougen, "Evolution of robot-to-robot nurturing and nurturability," in *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, Nov 2012, pp. 7 pages, unnumbered.

[14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, USA: MIT Press, 1998.

[15] J. A. P. J. Kober, Jens; Bagnell, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[16] W. B. Powell, "Ai, or and control theory: A rosetta stone for stochastic optimization," Princeton University, Tech. Rep., 2012.

[17] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, 2003, pp. 1–20.

[18] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[19] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, University of Massachusetts, 1984.

[20] D. F. Hougen, "Connectionist reinforcement learning for control of robotic systems," Ph.D. dissertation, University of Minnesota, 1998, aAI9907501.

[21] V. Gullapalli, "Reinforcement learning and its application to control," Ph.D. dissertation, University of Massachusetts, 1992.

[22] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, no. 1, pp. 45–83, Jul. 1991. [Online]. Available: https://link.springer.com/article/10.1023/A:1022619109594

[23] L. Zaval and T. M. Gureckis, "The impact of perceptual aliasing on exploration and learning in a dynamic decision making task," in *Proceedings of the Cognitive Science Society*, vol. 32, 2010. [Online]. Available: http://escholarship.org/uc/item/5169k6m2.pdf

[24] J. Suh and D. F. Hougen, "The context-aware learning model: reward-based and experience-based logistic regression backpropagation," in *IEEE Symposium Series on Computational Intelligence*. IEEE, 2017, p. 8 pages, to appear.

[25] S. N. H. Shah, "Nurturing promotes the evolution of learning in changing environments," Ph.D. dissertation, University of Oklahoma, Norman, OK, USA, 2015.

[26] D. Floreano and J. Urzelai, "Evolutionary robots with on-line self-organization and behavioral fitness," *Neural Networks*, vol. 13, no. 4-5, pp. 431–443, May 2000. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(00)00032-0

[27] Y. Niv, D. Joel, I. Meilijson, and E. Ruppin, "Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors," *Adaptive Behaviour*, vol. 10, no. 1, pp. 5–24, 2002. [Online]. Available: http://dx.doi.org/10.1177/10597123020101001

[28] M. Woehrer, D. F. Hougen, I. Schlupp, and B. E. Eskridge, "Robot-to-robot nurturing: A call to the research community," in *International Conference on Development and Learning and Epigenetic Robotics*, 2012, pp. 2 pages, unnumbered. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6384412

[29] M. Rothschild and J. Stiglitz, "Increasing risk: I. a definition," *Journal of Economic Theory*, vol. 2, no. 3, pp. 225–243, 1970. [Online]. Available: http://EconPapers.repec.org/RePEc:eee:jetheo:v:2:y:1970:i:3:p:225-243

[30] S. N. H. Shah and D. F. Hougen, "Nurturing promotes the evolution of reinforcement learning in changing environments," in *2017 IEEE Symposium on Computational Intelligence in Control and Automation (CICA)*. IEEE, 2017, to appear.