Improved Stochastic Synapse Reinforcement Learning for Continuous Actions in Sharply Changing Environments

Syed Naveed Hussain Shah[†]
MS Office
Microsoft Corporation
Redmond, WA, USA
sayyed.naveed@gmail.com
0000-0002-7418-870X

Dean Frederick Hougen
Dept. of Computer Science
University of Oklahoma
Norman, OK, USA
hougen@ou.edu
0000-0001-5393-1480

***Abstract*—Reinforcement learning in continuous action spaces requires mechanisms that allow for exploration of infinite possible actions. One challenging issue in such systems is the amount of exploration appropriate during learning. This issue is complicated further in sharply changing dynamic environments. Reinforcement learning in artificial neural networks with multiparameter distributions can address all aspects of these issues. However, which equations are most appropriate for updating these parameters remains an open question. Here we consider possible equations derived from two sources: The classic equations proposed for REINFORCE and modern equations introduced for Stochastic Synapse Reinforcement Learning (SSRL), as well as combinations thereof and variations thereon. Using a set of multidimensional robot inverse kinematics problems, we find that novel combinations of these equations outperform either set of equations alone in terms of both learning rate and consistency.**

I. Introduction

Reinforcement learning (RL) is an approach to machine learning (ML) in which agents form policies for action selection based on reward received from the environment given environmental state and action selected. While RL is one of the most commonly studied forms of ML, after supervised and unsupervised learning, most research on reinforcement learning has focused on static environments [1]. *Static environments*, those that don't change during the learning process, are more easily learned than *dynamic environments*, which do. This is particularly true for RL, in which an agent that is tasked with maximizing rewards accumulated over time needs to decide between exploration of new possibilities (which is appropriate when the environment is largely unknown) and exploitation of already acquired knowledge (which is appropriate when the environment is well known). If an originally unknown environment remains static for a long period of time, the agent may gain sufficient experience with it for a (gradual) shift from exploration to exploitation to be appropriate. However, if the environment then changes radically, the agent's previous knowledge becomes outdated and counterproductive because it suggests actions that are highly inappropriate for the new environment. Unfortunately, the agent typically has no way to know whether or to what degree the environment has changed—it merely observes, acts, and receives rewards—so it cannot simply discard all prior knowledge when it receives a low reward as the environment might be risky (in the formal sense relating to stochasticity [2]) rather than dynamic.

While there have been some investigations into approaches to RL in dynamic environments [3]–[6] much recent attention in RL has incorporated mechanisms such as experience replay [6]–[9] that are incompatible with dynamic environments.

The problem of dynamic environments is complicated when the action space is continuous. In such environments, the agent's possible actions are unlimited, so discrete-action approaches such as Q-learning cannot be used [10]. Instead, methods such as policy gradient must be used to adjust policies toward increasingly rewarding actions. This exacerbates the question of exploration vs. exploitation—how much should the agent adjust policy parameters to follow the gradient if the landscape underlying that gradient might radically shift at any time?

Finally, perceptual aliasing [11], which can happen when input states are discretizations of a continuous underlying world to which the agent has no direct access, can result in desired outputs that vary for the same perceived state even in static and deterministic environments.

Here we investigate these very difficult RL problems using variants on a connectionist artificial neural network (ANN) approach with multiparameter distributions. Such approaches are time-tested and fundamental mechanisms for learning in continuous action spaces. In addition, these mechanisms can be combined with additional mechanisms such as deep preprocessing layers to allow for learning from raw pixels; however, here we study these mechanisms in isolation, so as not to conflate the targets of the study with extraneous factors.

In particular, we start with the well known foundational approach of REINFORCE with multiparameter distributions [12] (not to be confused with the more commonly seen REINFORCE with uniparameter distributions, which is appropriate for discrete action spaces [12]) and the recent approach of Stochastic

[†]Supported by Microsoft Corporation.

Synapse Reinforcement Learning (SSRL) [13]. Both approaches control policy and degree of exploration through separate parameters: μ and σ of Gaussian distributions, respectively. Although REINFORCE with multiparameter distributions compares favorably to recent popular RL approaches in continuous action spaces [14], we observed that SSRL greatly outperforms REINFORCE in dynamic environments [15], in large part because SSRL places stochasticity in synapses (the connections between neural units) whereas REINFORCE places it in units (the computational elements of ANNs) [16]. However, in addition to the placement of stochasticity, REINFORCE and SSRL use different equations to calculate their μ and σ parameters. Here we investigate the role those equations, and variations on them, play in determining system performance in dynamic continuous-action environments, with and without perceptual aliasing.

To put these concepts in a familiar ANN structure, we'll consider all μ values to be uniformly 0, so that the value at each synapse is determined by its weight w plus stochastic noise ϵ times the input value x on that connection. Then we'll update w directly, which is equivalent to leaving w unchanged and updating μ.

Because of the advantage of placing stochasticity in the synapses, we here consider the variant of REINFORCE called *REINFORCE S* because it uses the classic REINFORCE equations for updating μ and σ, but places its sampling in synapses [16].

Because of the importance of appropriate levels of exploration in dynamic environments, we hypothesize that the σ-update equations of SSRL will play a key role in determining algorithm performance.

II. Approach

Here we consider only standard feed-forward, fully connected networks with summation units, logistic activation functions, and a single hidden layer [17]. Only the output layer of the network is trained using one of the RL alternatives described herein; any previous layers of the network are trained by backpropagating weight adjustments from the output layer through the hidden units.

While we here only consider a single hidden layer and backpropagating weight updates, these methods are compatible with networks of arbitrary depths and any form of gradient descent to update those layers. The use of this minimal ANN architecture is intentional as we wish to isolate effects of the learning rules themselves, which could be masked or amplified by introducing extraneous mechanisms to the experimental environment.

While both REINFORCE and SSRL have been described previously [12], [13], we re-describe them here for completeness and with slightly different notation to allow for uniform notation to be used for REINFORCE S, SSRL, and the variants of both. Note that all text and equations are common to all algorithms in question, unless specified otherwise.

To calculate the network's output at time τ, a pass through the network is made, starting from the input layer and working forward toward the output layer [17]. In the final layer, noise is sampled from a distribution associated with each synapse between the (last) hidden layer and the output layer, using

$$\epsilon_{kj}(\tau) \sim \Psi(\mu_{kj}(\tau), \sigma_{kj}(\tau)) \tag{1}$$

where k and j represent units in hidden and output layers, respectively, $\epsilon_{kj}(\tau)$ is the noise sampled from the normal distribution at synapse kj, $\mu_{kj}(\tau)$ is the mean of the noise distribution and $\sigma_{kj}(\tau)$ is the standard deviation at synapse kj, all at time τ.

The interior value $net_j(\tau)$ of each unit j in the output layer is computed using

$$net_j(\tau) = \frac{\sum_{k=1}^{K} y_k(w_{kj}(\tau) + \epsilon_{kj}(\tau))}{K}, \tag{2}$$

where K is the number of inputs from the hidden layer, w_{kj} is the synaptic weight between units k and j, and y_k is the output of unit k. The output $o_j(\tau)$ of each unit in the output layer, then, is computed using the logistic activation function

$$o_j(\tau) = \frac{1}{1 + e^{-\lambda net_j(\tau)}}. \tag{3}$$

Network outputs determine the (multidimensional) action for which reward $r(\tau)$ is assigned. Network parameters (weights w and exploration parameters σ) are adjusted, as follows.

For REINFORCE S, each weight difference $\delta_{w_{kj}}(\tau)$ for synaptic weight $w_{kj}(\tau)$ is computed using

$$\delta_{w_{kj}}(\tau) = (r(\tau) - \hat{r}(\tau))\epsilon_{kj}(\tau), \tag{4}$$

where $\hat{r}(\tau)$ is the expected reward.

For both REINFORCE S and SSRL, the expected reward used in this study is $\hat{r}(\tau+1) = d\,r(\tau) + (1-d)\,\hat{r}(\tau)$, where d is a discount factor. This calculation for expected reward comes from the original description of REINFORCE [12]. However, both REINFORCE S and SSRL are compatible with any calculation of expected reward.

For SSRL, the corresponding weight difference equation is

$$\delta_{w_{kj}}(\tau) = (r(\tau) - \hat{r}(\tau))y_k(\tau)\epsilon_{kj}(\tau), \tag{5}$$

where the additional term y_k for the output from unit k is included to account for the influence of that term on the output for which reward was received. That is, this term is intended to address the structural credit assignment problem [18]. Because the term y_k is multiplied by w_{kj} to form one of the inputs to unit j, we hereafter refer to it as the *input factor*, since our calculations are from the perspective of unit j.

Weight updates are then computed from these weight difference values using

$$w_{kj}(\tau+1) = w_{kj}(\tau) + \eta_w \delta_{w_{kj}}(\tau), \tag{6}$$

where η_w is the learning rate for weights between the hidden and output layers.

For REINFORCE S, $\delta_{\sigma_{kj}}(\tau)$, which is the amount of change in the exploration parameter σ at synapse kj, is computed using

$$\delta_{\sigma_{kj}}(\tau) = \frac{\epsilon_{kj}(\tau)^2 - \sigma_{kj}(\tau)^2}{\sigma_{kj}(\tau)}. \tag{7}$$

TABLE I
EQUATION COMBINATIONS. PREFIX SS OR RS DENOTES SSRL OR REINFORCE S, RESPECTIVELY. LOWERCASE i AND n DENOTE INPUT FACTOR OR NO INPUT FACTOR WHERE THE FIRST LETTER IN EACH COMBINATION RELATES TO SIGMA (σ) AND THE SECOND RELATES TO WEIGHT (w), RESPECTIVELY. NUMBERS IN ANGLE BRACKETS REFLECT RANKINGS BASED ON RESULTS.

		Sigma Update			
		Basis: SSRL (SS)		Basis: REINFORCE S (RS)	
		i (Eq. 8) (Input Factor)	n (Eq. 10) (No Input Factor)	i (Eq. 9) (Input Factor)	n (Eq. 7) (No Input Factor)
Weight Update	n (Eq. 4) (No Input Factor)	SSin $\langle 1 \rangle$	SSnn $\langle 2 \rangle$	RSin $\langle 5 \rangle$	RSnn $\langle 7 \rangle$ (REINFORCE S)
	i (Eq. 5) (Input Factor)	SSii $\langle 3 \rangle$ (SSRL)	SSni $\langle 4 \rangle$	RSii $\langle 6 \rangle$	RSni $\langle 8 \rangle$

For SSRL, the corresponding σ difference equation is

$$\delta_{\sigma_{kj}}(\tau) = y_k(\tau)(|\epsilon_{kj}(\tau)| - c_\sigma \sigma_{kj}(\tau)), \tag{8}$$

where constant c_σ inclines the algorithm toward exploration or exploitation. Again the SSRL equation includes the input factor $y_k(\tau)$ to help account for the influence of that term on the output at the final layer, whereas the REINFORCE S equation does not. In addition, calculations concerning the difference between the sampled noise ϵ and the standard deviation σ are quite different between the algorithms. The origin and importance of these differences are examined in the discussion.

To test our hypothesis as well as to analyze the importance of the input factor in the σ difference equation, we introduce variation of Equations 7 and 8 as follows.

In the REINFORCE S σ difference equation (Equation 7), we introduce the input factor, giving

$$\delta_{\sigma_{kj}}(\tau) = y_k(\tau)\frac{\epsilon_{kj}(\tau)^2 - \sigma_{kj}(\tau)^2}{\sigma_{kj}(\tau)}. \tag{9}$$

Conversely from the SSRL σ difference equation (Equation 8) we exclude the input factor, giving

$$\delta_{\sigma_{kj}}(\tau) = |\epsilon_{kj}(\tau)| - c_\sigma \sigma_{kj}(\tau). \tag{10}$$

Finally, the σ values themselves are updated using

$$\sigma_{kj}(\tau+1) = \sigma_{kj}(\tau) + \eta_\sigma(r(\tau) - \hat{r}(\tau))\delta_{\sigma_{kj}}(\tau), \tag{11}$$

where η_σ is the learning rate for σ values and subject to $\sigma \geq 0$. That is, if Equation 11 would result in a negative value for σ, σ is set equal to 0.

To formalize and test our hypotheses, we propose comparing all combinations of Equations 4 and 5 with 7, 8, 9, and 10 to empirically test the contribution of each component of each equation (e.g., the input factor) on system performance. These combinations are presented in Table I.

III. EXPERIMENTS

To facilitate comparisons with prior work, experiments use an existing environment [16]. This setup uses multiple non-linear data sets based on the inverse kinematics of a simplified model of a PUMA robotic arm with all values for all data sets scaled to be in $[0,\ 1]$. In each data set, each input-output vector pair is generated by randomly sampling joint values from a uniform distribution, calculating target outputs, i.e., generating location values, using underlying continuous functions from the forward kinematics model. Because these are inverse kinematics, input and output are then swapped, thus desired locations are inputs and corresponding joint values are targets. For discrete input, the resulting input values are discretized to provide perceptually aliased inputs and corresponding continuous valued output vectors. These data sets ensure that each input-output vector pair is almost certainly unique, and each pair is presented only once to the learning agent so it must be able to generalize in order to improve its performance over time.

To use this for RL, we tell a robot to place its end effector at a desired location and (in some cases) orientation, let the robot move its arm based on the output of its ANN, then calculate how close it's joint values are to those necessary to achieve the target position. That is, the network is presented with one input vector for each trial (learning experience) τ, then a scalar reward value $r(\tau)$ is based on the difference between the network-generated output vector and the target output vector from the training data on that trial, calculated as

$$r(\tau) = 1 - \frac{\sum_{j=1}^{J}|t_j - o_j|}{J}, \tag{12}$$

where J is the number of output units, t_j is the target at unit j, and o_j is the output, giving reward values in $[0,\ 1]$. (Similar RL problems have been standard for decades [19], [20].)

To make this environment dynamic, we sample the inputs from highly separated regions of the range of each joint during each half of training. This means that the learning agent must first learn the correct joint angles for reaching within one portion of its workspace, then quickly unlearn those angles and learn new joint angles for a disparate portion of its workspace based only on reward values (without ever receiving an explicit indication that the environment has changed).

The 6×6 inverse kinematics problem is to predict the values of the waist, shoulder, elbow, and three joints of a spherical wrist, given the position in Cartesian three-space (x, y, z) and orientation (given as roll, pitch, and yaw) of the end effector. The 3×3 data sets use only the waist, shoulder, and elbow joints with the position in three-space. The 6×3 data sets use the waist, shoulder, and elbow joints with position and orientation.

To also investigate perceptual aliasing [11], additional data sets are created by discretizing input to two or three digits of precision, using seven or ten bits, for a total of 21 or 30 binary inputs, respectively. For location plus orientation, values are discretized to one digit of precision using four bits each for a total of 24 binary inputs. Thus, independent 21×3, 30×3, and 24×6 data sets are generated with perceptually aliased input-output mappings.

For each data set, 20000 input-output vector pairs are generated. The first and second halves of each data set are generated using contrasting ranges of joints, causing it to reach to very different positions during the first and second halves of each run, resulting in the desired sharply changing environment. 50 such data sets are used for each experimental condition to collect statistically meaningful results. Note that these non-linear data sets are moderately high dimensional and are on-par with other continuous action data sets used for RL experiments, such as the robotics and control environments from OpenAiGym (https://gym.openai.com/). However, ours have been made more difficult by making them dynamic.

Each input unit has a bias input, the hidden layer has units equal to the number of input units plus one, the initial weights are in $[-2, +2]$, $\lambda = 2$ for the logistic functions, all learning rates and discount factors are 0.5 (a moderate value), and σ is initialized uniformly randomly in $[0,\ 1]$ and kept non-negative as suggested for REINFORCE [12].

IV. Results

Figure 1 shows the average reward collected per trial by all variants across all runs. Figures 1a, 1c, and 1e show three input, three output (3×3), six input, three output (6×3), and six input, six output (6×6) continuous state-action cases respectively. Figures 1b, 1d, and 1f, show twenty one input, three output (21×3), thirty input, three output (30×3), and twenty four input, six output (24×6) discrete state and continuous action cases respectively. The apparent differences between these curves within each data set are statistically significant (randomized ANOVA, $p < 0.001$ for both algorithm and interaction for all experiments and all pairwise comparisons [21]).

Figure 2 shows a set of box and whisker plots illustrating distributions of cumulative rewards collected for each of 50 independent runs for all algorithm variations. The results are summarized in Table II. The order of the sub-figures is the same as for Figure 1. Here, within each individual sub-figure, the eight boxes on the left side represent the cumulative rewards collected during the entire run, the eight in the middle represent the cumulative reward in the last 10% of the trials before change when the algorithms have had time to converge, while the eight on the right side represent the cumulative reward in the last 10% of the second half of the trials (after the sharp change is introduced mid-way at trial 10000).

The results are statistically significant (Quade test, $p < 0.05$ for both pre- and post-hoc statistics for all experiments and all comparisons after the sharp change, $d.f._N = 7$, $d.f._D = 343$, [22]) except a small percentage of post-hoc comparisons, as follows. For SSni compared to RSin, cumulative reward distributions are not significantly different for 3×3 ($p = 0.367$) and 6×6 ($p = 0.402$). For RSnn compared to RSni, the distributions are not significantly different for 6×3 ($p = 0.421$) and 6×6 ($p = 0.062$). For SSnn compared to SSin, the distributions are not significantly different for 3×3 ($p = 0.118$), 21×3 ($p = 0.066$), and 30×3 ($p = 0.139$). For SSni compared to SSii, the distributions are not significantly different for 21×3 ($p = 0.181$) and 24×6 ($p = 0.589$). For SSnn compared to SSii, the distributions are not significantly different for 6×3 ($p = 0.202$). Finally, SSii compared to RSin, the distributions are not significantly different for 30×3 ($p = 0.750$).

Table II shows the average of the average reward collected for all variants on all data sets as well as the average of the standard deviation of the reward collected on all data sets for the entire run, the last 10% of trials before the change, and the last 10% of trials after the change.

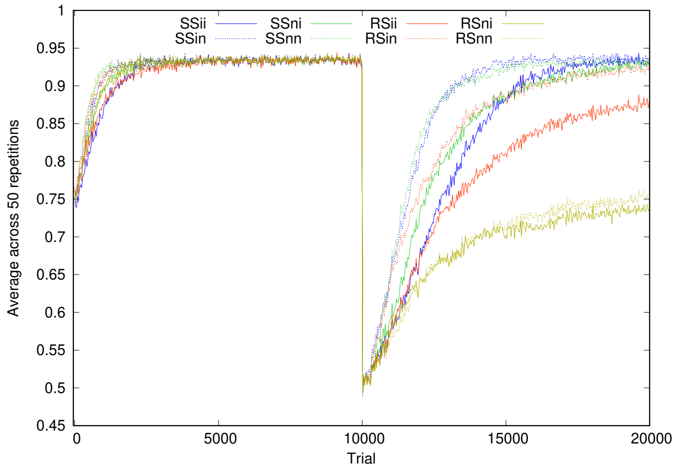
V. Discussion

Overall, these results clearly indicate that novel equation sets SSin and SSnn, introduced here by using the basic σ update equation from SSRL (with or without the input factor) along with the weight update equation from REINFORCE S (that is, without the input factor), consistently outperform all other equations both in terms of fast learning and higher cumulative rewards after the dynamic change. Between the two, SSin outperforms its counterpart SSnn in more data sets indicating that input multiplication in the σ update equation generally improves learning speed and reward accumulation when the algorithm must adapt after a change. Furthermore, SSin also consistently exhibits the lowest standard deviation, in the end, for the cumulative reward collected across 50 runs. Combined, these results show that the novel SSin equation set introduced here manifests the best relative exploratory and exploitatory policy learning capabilities.

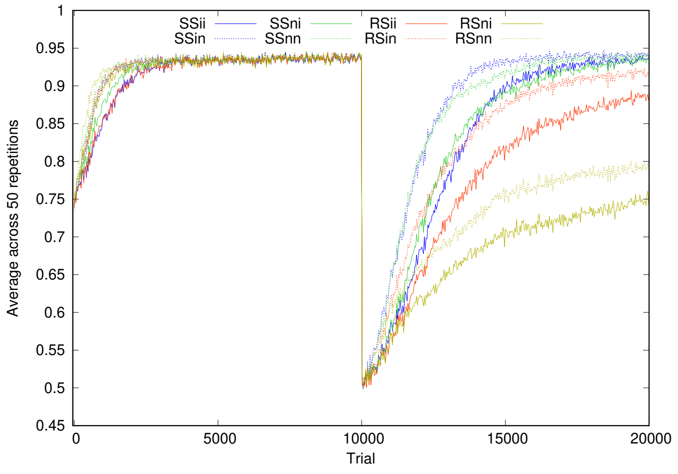
In addition, it is interesting to note that including the input factor in the σ update equation from REINFORCE S (yielding novel equation set RSin) substantially improves the original algorithm's (REINFORCE S's) performance after the change.

Ranking performance, SSin and SSnn achieve similar near-optimal performance in the final 10% of trials both before and after the change. Interestingly, novel equation set SSin achieves equal or higher cumulative reward after the change compared to before in all but one case. Conversely, SSnn, also a novel equation set, performs relatively better before the change. This suggests SSin is better suited for sharply changing environments. SSii (the original SSRL) has the next highest cumulative rewards and almost matches the performance of the top two by the end, albeit exhibiting relatively slower initial learning post change. Both SSni and RSin perform head-on, thus clear superiority is not obvious. SSni's total sum of cumulative reward for all data sets, after change, during the last 10% of the trials is 552.4 with an average of 92.06 and sum of the standard deviation is 15.9 with an average of 2.65. Corresponding numbers for RSin are a sum of 552.2 with an average of 92.03 and sum of standard deviation of 18.1 with an average of 3.01. This perhaps makes SSni marginally better than RSin. Finally, RSii shows superior performance to RSnn (REINFORCE S) which performs better than RSni.

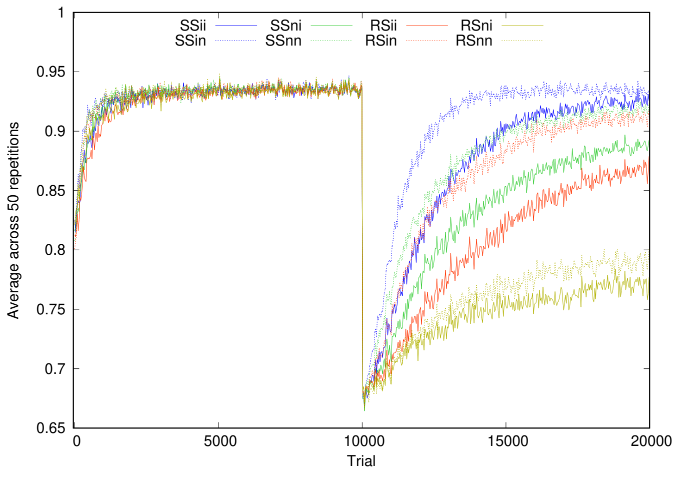
The variants using the core σ update equation from SSRL clearly outperform the variants using the core σ update equation



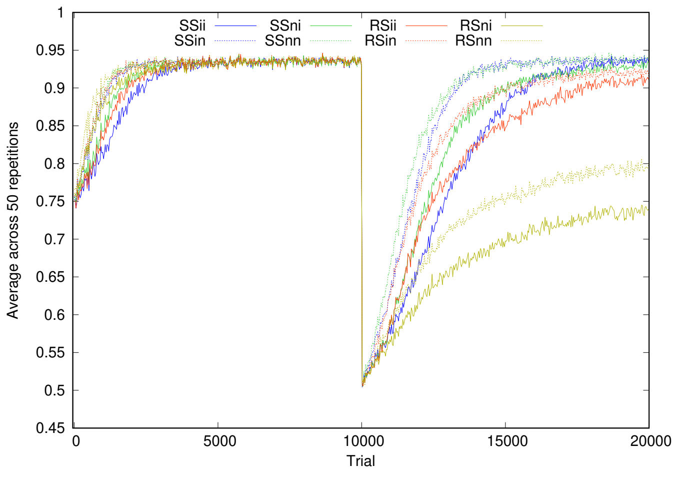
(a) 3×3



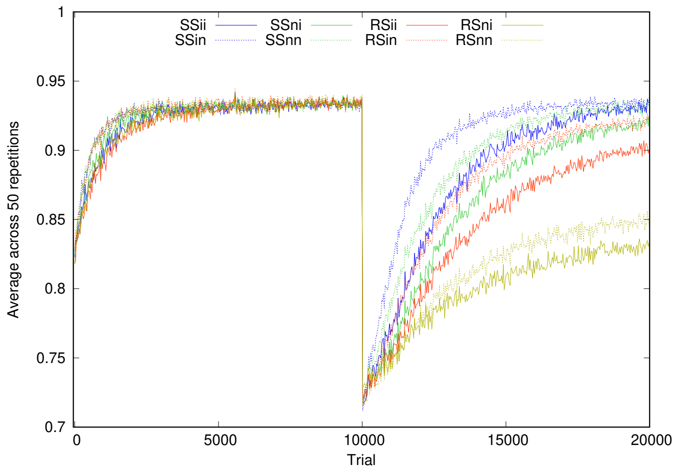
(b) 21×3



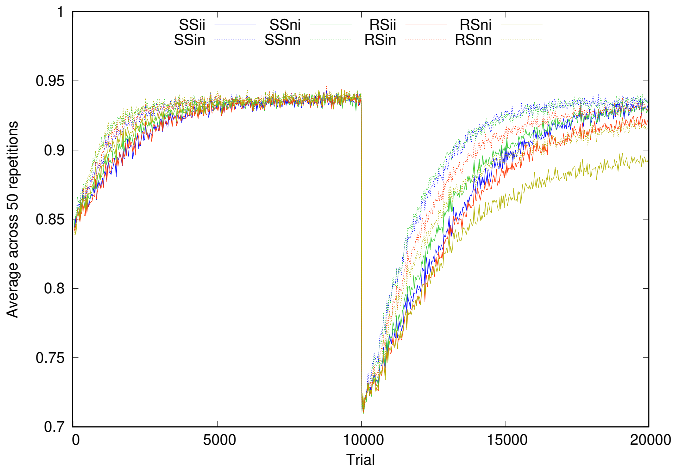
(c) 6×3



(d) 30×3



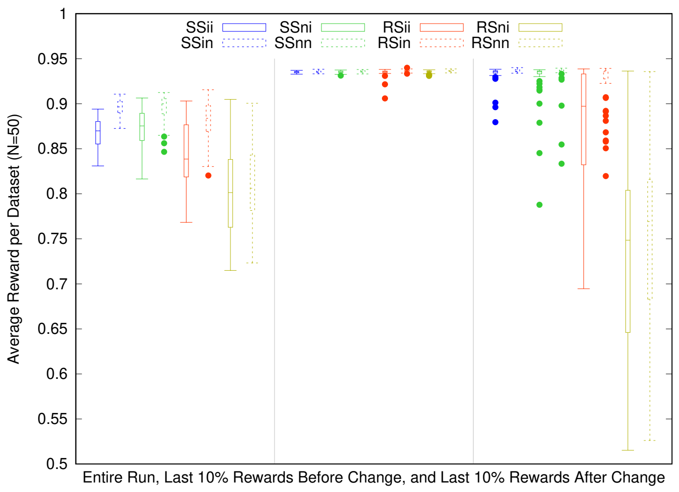
(e) 6×6



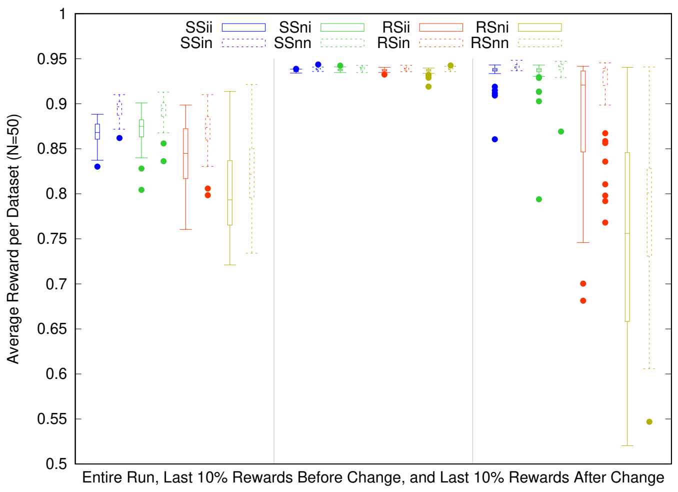
(f) 24×6

Fig. 1. Average performance curves. Average reward collected per trial across 50 runs by all variants. The drop in the performance at trial 10000 is where the sharp change occurs.

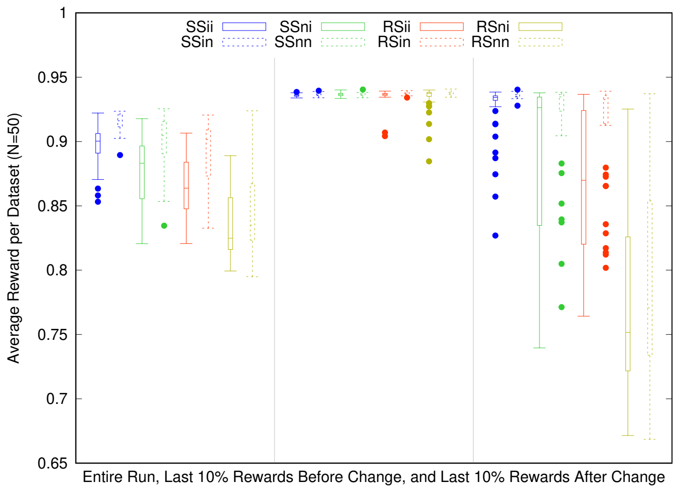
from REINFORCE S, with the only exception being RSin competing closely with SSni where RSin shows substantial improvement over REINFORCE S. Finally, within each of the two groups (i.e., SSRL-based σ update and REINFORCE S-based σ update), the input factor in σ update and no input factor in the weight update equations (see SSin and RSin)



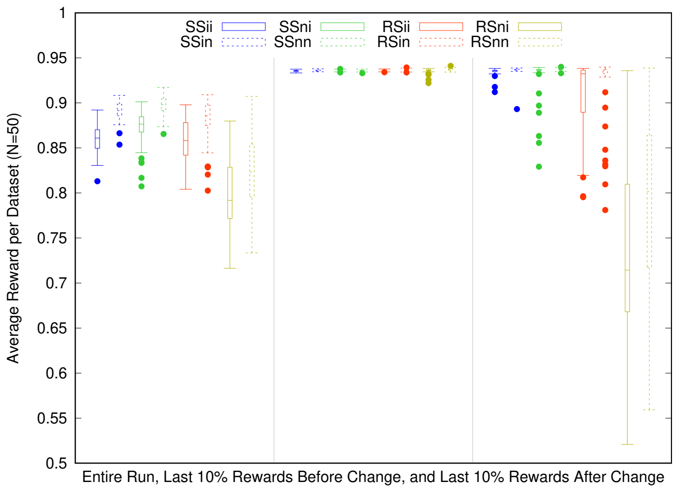
(a) 3×3



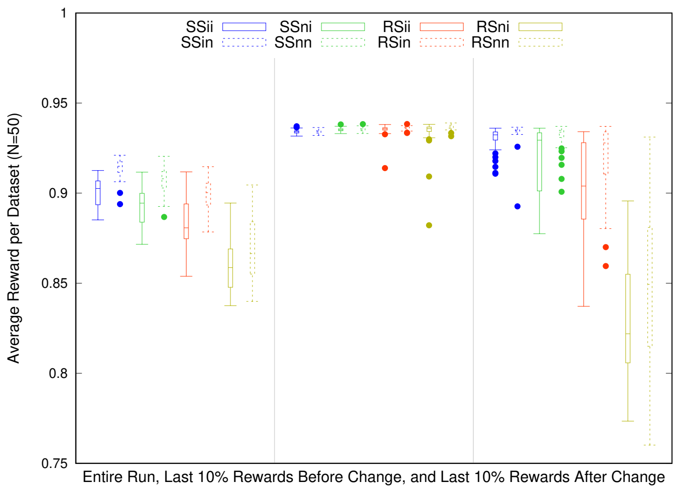
(b) 21×3



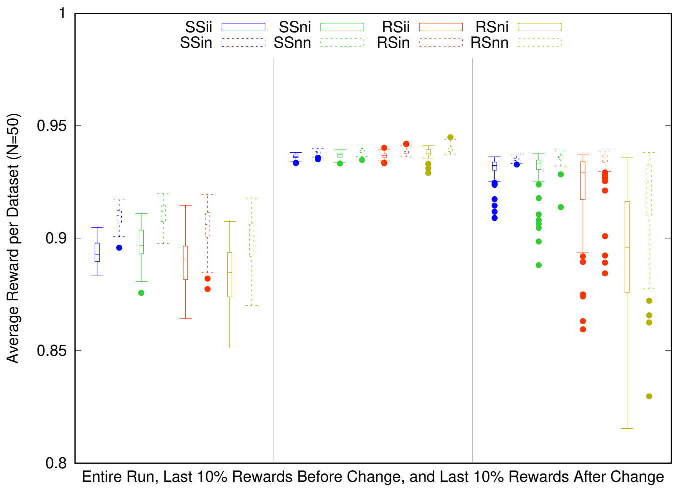
(c) 6×3



(d) 30×3



(e) 6×6



(f) 24×6

Fig. 2. Average cumulative reward performance. Box and whisker plots with outliers of average cumulative reward per run across 50 runs by all variants. First one-third in each plot shows average cumulative reward received over entire run while the second one-third shows the same for the last 10% of trials before change and the last one-third shows the last 10% of trials before the end of the run.

TABLE II

AVERAGES OF AVERAGE REWARDS ($\overline{\overline{r}}$) AND AVERAGES OF STANDARD DEVIATIONS OF REWARDS ($\overline{\sigma_r}$) ACROSS ALL RUNS. TABLE DEPICTS AVERAGES FOR THE ENTIRE RUN (TOP), LAST 10% OF THE TRIALS BEFORE CHANGE (MIDDLE), AND LAST 10% OF THE SECOND HALF'S TRIALS OF EACH RUN (BOTTOM). NUMBERS UNDERLINED ARE THE HIGHEST REWARDS AND LOWEST STANDARD DEVIATIONS.

		SSii		SSin		SSni		SSnn		RSii		RSin		RSni		RSnn	
	Data set	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$
Entire Run	3×3	86.6 %	1.7 %	$\underline{89.6\%}$	$\underline{0.9\%}$	87.4 %	2.2 %	$\underline{89.6\%}$	1.5 %	84.4 %	3.5 %	88.0 %	2.4 %	80.1 %	5.2 %	80.7 %	4.5 %
	6×3	89.7 %	1.5 %	$\underline{91.5\%}$	$\underline{0.6\%}$	87.8 %	2.7 %	90.0 %	2.0 %	86.4 %	2.2 %	89.4 %	2.1 %	83.5 %	2.5 %	84.5 %	3.1 %
	6×6	90.0 %	0.7 %	$\underline{91.4\%}$	$\underline{0.5\%}$	89.3 %	1.0 %	90.7 %	0.7 %	88.3 %	1.4 %	89.9 %	0.8 %	86.0 %	1.5 %	86.9 %	1.7 %
	21×3	86.7 %	1.3 %	$\underline{89.4\%}$	$\underline{1.1\%}$	87.2 %	1.7 %	89.2 %	1.5 %	84.1 %	3.3 %	87.1 %	2.4 %	79.8 %	5.0 %	82.4 %	4.5 %
	30×3	86.0 %	1.6 %	89.1 %	$\underline{1.0\%}$	87.2 %	2.0 %	$\underline{89.6\%}$	1.1 %	85.8 %	2.4 %	88.0 %	2.4 %	79.5 %	4.0 %	82.4 %	3.9 %
	24×6	89.3 %	0.5 %	91.0 %	$\underline{0.4\%}$	89.7 %	0.8 %	$\underline{91.1\%}$	0.5 %	88.9 %	1.1 %	90.5 %	0.9 %	88.3 %	1.3 %	89.9 %	1.1 %
Mid-last 10% trials	3×3	93.5 %	$\underline{0.1\%}$	93.5 %	$\underline{0.1\%}$	93.5 %	$\underline{0.1\%}$	$\underline{93.6\%}$	$\underline{0.1\%}$	93.5 %	0.5 %	$\underline{93.6\%}$	$\underline{0.1\%}$	$\underline{93.6\%}$	0.2 %	$\underline{93.6\%}$	$\underline{0.1\%}$
	6×3	93.6 %	$\underline{0.1\%}$	93.6 %	$\underline{0.1\%}$	$\underline{93.7\%}$	$\underline{0.1\%}$	$\underline{93.7\%}$	$\underline{0.1\%}$	93.6 %	0.6 %	$\underline{93.7\%}$	$\underline{0.1\%}$	93.4 %	1.0 %	$\underline{93.7\%}$	$\underline{0.1\%}$
	6×6	93.4 %	$\underline{0.1\%}$	93.4 %	$\underline{0.1\%}$	93.5 %	$\underline{0.1\%}$	93.5 %	$\underline{0.1\%}$	93.5 %	0.3 %	93.6 %	$\underline{0.1\%}$	93.4 %	0.9 %	$\underline{93.7\%}$	$\underline{0.1\%}$
	21×3	93.6 %	$\underline{0.1\%}$	93.8 %	$\underline{0.1\%}$	93.8 %	$\underline{0.1\%}$	$\underline{93.9\%}$	0.2 %	93.7 %	$\underline{0.1\%}$	$\underline{93.9\%}$	0.2 %	93.7 %	0.3 %	$\underline{93.9\%}$	$\underline{0.1\%}$
	30×3	93.5 %	$\underline{0.1\%}$	93.6 %	$\underline{0.1\%}$	93.6 %	$\underline{0.1\%}$	93.6 %	$\underline{0.1\%}$	93.6 %	$\underline{0.1\%}$	$\underline{93.7\%}$	$\underline{0.1\%}$	93.6 %	0.3 %	$\underline{93.7\%}$	$\underline{0.1\%}$
	24×6	93.6 %	$\underline{0.1\%}$	93.8 %	$\underline{0.1\%}$	93.7 %	$\underline{0.1\%}$	93.9 %	$\underline{0.1\%}$	93.7 %	$\underline{0.1\%}$	93.9 %	$\underline{0.1\%}$	93.8 %	0.2 %	$\underline{94.0\%}$	0.2 %
Last 10% trials	3×3	93.3 %	1.1 %	$\underline{93.7\%}$	$\underline{0.1\%}$	92.7 %	2.6 %	93.2 %	1.9 %	87.5 %	6.8 %	92.2 %	2.8 %	73.7 %	12.5 %	75.1 %	10.9 %
	6×3	92.6 %	2.2 %	$\underline{93.6\%}$	$\underline{0.2\%}$	88.8 %	5.9 %	91.9 %	3.6 %	86.8 %	5.6 %	91.2 %	4.0 %	77.1 %	6.4 %	79.0 %	8.0 %
	6×6	93.0 %	$\underline{0.6\%}$	$\underline{93.4\%}$	$\underline{0.6\%}$	91.8 %	1.9 %	93.1 %	0.7 %	90.1 %	2.8 %	92.0 %	1.8 %	83.0 %	3.4 %	84.8 %	4.6 %
	21×3	93.5 %	1.3 %	$\underline{94.1\%}$	$\underline{0.3\%}$	93.3 %	2.1 %	93.9 %	1.1 %	88.4 %	6.7 %	91.7 %	4.4 %	74.5 %	12.4 %	79.0 %	9.9 %
	30×3	93.5 %	0.4 %	93.6 %	0.6 %	92.9 %	2.3 %	$\underline{93.7\%}$	$\underline{0.1\%}$	90.9 %	4.2 %	92.0 %	3.9 %	73.7 %	11.3 %	79.3 %	9.4 %
	24×6	93.0 %	0.6 %	$\underline{93.5\%}$	$\underline{0.1\%}$	92.9 %	1.1 %	$\underline{93.5\%}$	0.4 %	92.1 %	2.0 %	93.1 %	1.2 %	89.1 %	3.1 %	91.7 %	2.3 %

produced the best performers within their respective groups. Conversely, no input factor in the σ update equation along with including the input factor in weight update equations produced the worst performers within each group (see SSni and RSni). This clearly suggests an important result that inputs from the previous layer contribute substantially to the benefit of exploration/exploitation trade-off while ignoring the inputs helps adjust weights appropriately. In a nutshell, this affirms the need to address the structural credit assignment problem while formulating the σ update equations.

Interestingly, the core σ update equation from REINFORCE S is inherited from REINFORCE with multiparameter distributions [12], where the characteristic eligibility for the change in σ is calculated as the partial derivative with respect to σ of a Gaussian density function (the function from which the noise values are sampled). This makes sense, as the objective is to follow the gradient not only in weight space to update the policy, but also in σ space to update the degree of exploration. This is in contrast to the equation from SSRL, which was formulated intuitively based on direction of change rather than slope [13]. So, has intuition outperformed theory in this case? If so, how should equations be derived?

Then again, multiparameter REINFORCE does not use this calculation for σ directly. Instead, Williams says that a "reasonable algorithm is obtained by" multiplying the result of his partial derivative by σ^2. So, what does it tell us when this theoretical approach to deriving the equations does not produce "reasonable" algorithms on its own and their own author suggests substantial modifications to them?

Perhaps some of these questions could be addressed by evolving the equations [23], as suggested for future work.

VI. CONCLUSIONS

This empirical study systematically compares equations from REINFORCE S [16], an improved variant of classic REINFORCE algorithm with multiparameter distributions [12], and Stochastic Synapse Reinforcement Learning (SSRL) [13]. The study focuses on comparing and analyzing the role of the core equations for updating the exploration parameter σ as well as the input factor y_k in weight and σ update equations of both algorithms. Alternate equations are hypothesized, formulated, and then empirically compared against each other using PUMA robotic arm inverse kinematics with moderately high dimensional data sets using both continuous and discrete aliased states and continuous actions spaces. Understanding and improving update equations for σ is shown to be key to high performance in sharply changing dynamic environments—a largely overlooked area of research. Moreover, it appears that concentrating on the direction of the updates to σ is more effective than using information on slope.

The results strongly suggest including the input factor in the σ update equations while omitting the input factor from the weight update equations, thus indicating that taking into account the structural credit assignment is not as simple as including the same factors in all equations. Both algorithms find better sets of alternate weight and σ update equations that follow the aforementioned rules. The statistical hypothesis tests, randomized ANOVA and Quade, reflect the best algorithms to be SSin in SSRL's variants group and RSin in the REINFORCE S group. The results and tests further reveal that SSin outperforms all other algorithms with statistical significance. Furthermore, SSin performs well in both continuous and discrete aliased state spaces with the later being known for the additional perceptual aliasing problem. The high cumulative reward and low reward variability by SSin reflect its superior exploratory and exploitatory policy learning capabilities following stochastic policy gradients.

Ranking the algorithms from best to worst performance follows as: SSin $>$ SSnn $>$ SSii (SSRL) $>$ SSni $\geq$ RSin $>$ RSii $>$ RSnn (REINFORCE S) $>$ RSni, as reflected in Table I. The algorithms using SSRL's core σ update equation clearly outperform those using the core σ update equation from REINFORCE S (inherited from the classic REINFORCE). The results also suggest that the input factor aides with superior exploration/exploitation trade-off as long as the weight update equation is free from such multiplication.

VII. Future Work

A key takeaway of this study is that investigating alternatives for update equations can help find better algorithms. Using evolution of reinforcement learning to explore further variations of the algorithms and analyzing sensitivity of various parameters and hyper-parameters within the equations can help find even better algorithms that are faster learners and near optimal performers in dynamic environments [23], [24]. At the macro level, this could address some of the questions raised in the discussion with regard to how such equations should be derived. At the micro level, in this study, the sigmoidal activation function uses a λ value of 2 which extends the range of a unit's output values; however, evolving better λ values should be considered in the future. Interestingly, many traditional (non-evolved) RL methods are unsuited to dealing with change [6]–[9] whereas RL mechanisms evolved in dynamic environments are very unlikely to have this problem [23].

Using the most successful equations from this study on gradually changing, deep-learning, and/or delayed-RL tasks with temporal and/or structural credit assignment problems, and/or in modular or hierarchical RL, should improve understanding and highlight any performance and/or scalability issues.

However, while these empirical approaches can determine better performing algorithms, theoretical work is still needed to understand the reasons for these differences.

Finally, multiparameter REINFORCE [12] is mostly regarded as a goto algorithm for continuous actions spaces in the class of stochastic policy gradient algorithms [14]; however, REINFORCE with uniparameter distributions is not known to outperform the traditional Q-learning and SARSA in discrete action spaces. Future studies investigating the update equations underlying these algorithms can perhaps improve on the state of the art in discrete action spaces as well.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/RLbook2018.pdf

[2] M. Rothschild and J. E. Stiglitz, "Increasing risk I: A definition," *Journal of Economic Theory*, pp. 225–243, 1970.

[3] E. D. S. Costa and M. M. Gouvea Jr., "Autonomous navigation in dynamic environments with reinforcement learning and heuristic," in *International Conference on Machine Learning and Applications*, Dec. 2010, pp. 37–42.

[4] A. Mimura, S. Sumino, and S. Kato, "Adaptive reinforcement learning for dynamic environment based on behavioral habit," in *Intelligent Decision Technologies*. Springer Berlin Heidelberg, 2012, pp. 201–211.

[5] S.-Y. Chen, Y. Yu, Q. Da, J. Tan, H.-K. Huang, and H.-H. Tang, "Stabilizing reinforcement learning in dynamic environment with application to online recommendation," in *International Conference on Knowledge Discovery & Data Mining*. London, United Kingdom: ACM, 2018, pp. 1187–1196.

[6] L. Zhang, S. Lu, and Z.-H. Zhou, "Adaptive online learning in dynamic environments," *arXiv:1810.10815v1 [cs.LG]*, Oct. 2018. [Online]. Available: https://arxiv.org/abs/1810.10815v1

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, Sep. 2015, arXiv: 1509.02971. [Online]. Available: http://arxiv.org/abs/1509.02971

[9] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," *arXiv:1603.00748 [cs]*, Mar. 2016. [Online]. Available: http://arxiv.org/abs/1603.00748

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[11] 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.

[12] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992.

[13] S. N. H. Shah and D. F. Hougen, "Stochastic synapse reinforcement learning (SSRL)," in *IEEE Symposium Series on Computational Intelligence*, Nov 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SSCI.2017.8285425

[14] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[15] S. N. H. Shah and D. F. Hougen, "Stochastic reinforcement learning for continuous actions in dynamic environments," in *Florida Artificial Intelligence Research Society (FLAIRS) International Conference*, May 2020, p. 6 pages.

[16] ——, "Rethinking stochasticity in neural networks for reinforcement learning with continuous actions," in *IEEE Symposium Series on Computational Intelligence*, Dec 2019, pp. 487–495. [Online]. Available: https://doi.org/10.1109/SSCI44817.2019.9002826

[17] A. P. Engelbrecht, *Computational Intelligence: An Introduction*, 2nd ed. Wiley, 2007.

[18] H. Saito, K. Katahira, K. Okanoya, and M. Okada, "Statistical mechanics of structural and temporal credit assignment effects on learning in neural networks," *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, vol. 83, no. 5 Pt 1, pp. 051 125–1–051 125–7, May 2011.

[19] H. Ritter and K. Schulten, "Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements," in *Neural Computers*, R. Eckmiller and C. v. d. Malsberg, Eds. Heidelberg: Springer, 1987, vol. F41, pp. 393–406.

[20] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural networks*, vol. 3, no. 6, pp. 671–692, 1990.

[21] J. H. Piater, P. R. Cohen, X. Zhang, and M. Atighetchi, "A randomized ANOVA procedure for comparing performance curves," in *Proceedings of the International Conference on Machine Learning*, vol. 98, 1998, pp. 430–438.

[22] D. Quade, "Using weighted rankings in the analysis of complete blocks with additive block effects," *Journal of the American Statistical Association*, vol. 74, no. 367, pp. 680–683, 1979.

[23] D. F. Hougen and S. N. H. Shah, "The evolution of reinforcement learning," in *IEEE Symposium Series on Computational Intelligence*, Dec 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SSCI44817.2019.9003146

[24] S. N. H. Shah and D. F. Hougen, "Nurturing promotes the evolution of reinforcement learning in changing environments," in *IEEE Symposium Series on Computational Intelligence*, 2017, pp. 1–8. [Online]. Available: https://doi.org/10.1109/SSCI.2017.8285400