The Thirty-Third International
FLAIRS Conference (FLAIRS-33)

Stochastic Reinforcement Learning for Continuous Actions in Dynamic Environments[*]

Syed Naveed Hussain Shah,[1] **Dean Frederick Hougen**[2]
[1]Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399,
[2]School of Computer Science, Gallogly College of Engineering,
University of Oklahoma, Norman, OK 73019–3009
sayyed.naveed@gmail.com, hougen@ou.edu

Abstract

Reinforcement learning (RL) agents use trial and error to learn action policies for environment states. Environments with continuous action spaces are far more challenging for RL than those with discrete actions because there are infinite possible continuous action values from which to choose. Dynamic environments create additional challenges for RL agents, which must adjust rapidly to changes. We recently introduced REINFORCE SUN, a superclass of REINFORCE with Gaussian units, that allows for stochasticity at different levels of granularity in artificial neural networks (synapse, unit, or network), and have shown that moving stochasticity to synapses greatly aids RL in both static and dynamic environments with continuous action spaces. However, we also found that performance in dynamic environments remained substantially lower than desired. To rectify this, we here consider alternative parameter update equations for learning in dynamic environments. These equations form the core of Stochastic Synapse Reinforcement Learning (SSRL), which we here generalize to create S*RL, a superclass of SSRL that allows for stochasticity at these levels. Empirical results using multi-dimensional robot inverse kinematic data sets show that S*RL update equations greatly outperform traditional REINFORCE equations in dynamic, continuous state and action spaces.

Introduction

Reinforcement learning (RL) is one of the most widely studied types of machine learning (Sutton and Barto 2018). An RL agent attempts to formulate a policy mapping perceived states to actions in order to maximize the reward gained from the environment over time, which results in a self-learning algorithm with important real-world applications. RL is particularly challenging because the agent must balance exploring the environment and possible policies, hopefully increasing its knowledge, with exploiting what it has already learned by taking the best known actions to try to maximize the total reward it receives, which is generally discounted over time.

Classic RL algorithms including TD (Sutton and Barto 1981) and Q-learning (Watkins 1989) have focused on discrete actions. Today it is still common to focus on RL with discrete actions and largely ignore continuous spaces because adapting popular RL algorithms to work in continuous spaces is difficult (Mnih et al. 2016). Nonetheless, there has been substantial research on RL using continuous actions, particularly for robotics applications (Deisenroth, Neumann, and Peters 2011), including some recent work (Heess et al. 2015; Duan et al. 2016; Lillicrap et al. 2019; Mao et al. 2019).

Continuous action RL is harder due to the infinite possible output values and consequent need to approximate the policy. To learn in such environments, policy gradient approaches are typically used (Heess et al. 2015; Schulman et al. 2018; Lillicrap et al. 2019; Mao et al. 2019). These methods sample continuous values for the agent's output and follow gradients based on rewards received. These approaches are generally paired with artificial neural networks (ANNs) to provide policy approximation that is often perturbed by stochastic noise to provide exploration. While REINFORCE with Gaussian stochastic units was the first and simplest of these methods (Williams 1992), it continues to provide surprisingly competitive results (Duan et al. 2016), is the de facto benchmark standard (Sutton and Barto 2018), and forms the core of many modern systems (Mnih et al. 2016).

Despite its strong and durable success, it is possible to substantially improve on the REINFORCE framework and, by extension, on numerous REINFORCE-based systems. We recently reconsidered the role of stochasticity in ANN-based RL agents, arguing that placing stochasticity in the synapses between the penultimate layer of an ANN and its output layer provides more granular and therefore more appropriate exploration/exploitation trade-offs than placing it in output units (as is traditional); introduced REINFORCE SUN, a superclass of REINFORCE that allows sampling to occur at different levels of granularity (synapse, unit, or network); and empirically demonstrated that placing sampling in synapses greatly improves performance (Shah and Hougen 2019).

Notwithstanding the excellent advances of synapse-based REINFORCE over unit-based REINFORCE, its performance in dynamic environments is less than desired. Learning in dynamic (non-stationary) environments is more difficult than learning in static environments (Sutton and Barto 2018). This is particularly true of environments that remain stationary for an extended period of time, then suddenly change, because

[*]Supported by Microsoft Corporation.

they require an agent that may have settled into exploiting to shift to exploration if it is to deal with its new environment.

Fortunately, there is an alternative: Stochastic Synapse Reinforcement Learning (SSRL), which is designed for dynamic environments (Shah and Hougen 2017). While there are other synapse-based RL approaches for continuous spaces, these were not designed for use in dynamic environments (Ma and Likharev 2006; Amravati et al. 2018).[1] In the present paper we introduce a superclass of SSRL that we term *S*RL* that, like REINFORCE SUN, allows for stochasticity at any level and compare it to REINFORCE SUN on dynamic problems.

Hypothesis

While we substantially improved on REINFORCE with Gaussian units by moving stochasticity to synapses (Shah and Hougen 2019), all versions of Gaussian REINFORCE struggle with learning in sharply changing dynamic environments. Hence, we here compare the performance of equations from an algorithm intended for dynamic environments: Stochastic Synapse Reinforcement Learning (SSRL) (Shah and Hougen 2017), yielding the following hypothesis:

H$_1$ *SSRL equations will outperform REINFORCE equations in dynamic environments.*

As with REINFORCE SUN, we recognize that it is possible to adapt the equations from SSRL to the unit and network level. Therefore, we propose a superclass of SSRL that we term *S*RL* with the wild-card "star" indicating the placement of the stochasticity, giving us not only SSRL, but also *SURL* and *SNRL*. Stochasticity at the synapse level is likely to perform better for S*RL, as it did for REINFORCE SUN, though this is not guaranteed, which gives our second hypothesis:

H$_2$ *SSRL will outperform SURL which will, in turn, outperform SNRL in both static and dynamic environments.*

Testing these hypotheses improves understanding of RL with continuous actions in dynamic environments. We test these hypotheses together in the environment in which we tested REINFORCE SUN (Shah and Hougen 2019).

Approach

A typical approach to generating continuous actions is to use an artificial neural network (ANN) as a policy approximator. However, typical ANNs are deterministic whereas RL requires exploration. Including stochastic noise provides exploration of the infinite continuous actions available.

REINFORCE with Gaussian units learns the standard deviation σ of a sampling distribution for each output unit in the ANN (Williams 1992). This allows units to learn the degree to which they should be exploratory. SSRL is conceptually similar to REINFORCE, but the equations used to update the parameters are notably different, as detailed in the following subsections, and noise is added to synapses, rather than units.

In ANNs, there is no reason for being equally exploratory with respect to each input to a given unit. By using Gaussian synapses, synapses can learn their own σ values. Conversely, given that the outputs from all units in the output layer contribute to the total reinforcement earned, we could consider a Gaussian network with a single σ value.

[1]These were motivated by ease of implementation in hardware.

The choice of update equations and noise placement gives REINFORCE S, U, or N, and SSRL, SURL, or SNRL, determined by whether the parameters are updated using REINFORCE SUN or S*RL equations and whether there is one σ and one corresponding noise sample each time step per synapse, unit, or network, respectively.

To put these concepts in a familiar ANN structure, we'll consider all sampling mean 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 y on that connection.

Here we consider only feedforward, fully connected ANNs with summation units and logistic activation functions and one hidden layer (Engelbrecht 2007). Only the final weight layer is updated using one of the RL alternatives described herein; previous weights are trained by backpropagation.[2]

General Learning Process

At each time step τ, the agent's state is fed as an input vector to the ANN, which propagates weighted activation values forward through its layers as in a typical feedforward ANN (Engelbrecht 2007), to its penultimate layer (the last layer of units before the output layer)[3]. At this point, noise is added to the computations to provide for exploration. This noise is included in calculating the output values of the network, which are used as the vector of continuous actions taken by the agent. The agent then receives reward from the environment, which is used to update both the weights of the ANN and the σ values that control exploration. The weight adjustments from the final weight layer are then backpropagated through the preceding layers of the network, as with a typical ANN.

It is in the addition of noise $\epsilon(\tau)$, and the updates of the synaptic weights $w(\tau)$ and the exploration parameter(s) $\sigma(\tau)$, that the various approaches described herein diverge.

Noise Computations

For network-based approaches (REINFORCE N and SNRL), at each time step τ, a noise value $\epsilon_N(\tau)$ is sampled for the entire network from a zero-mean Gaussian. Conceptually, this noise is added to each dimension of the action vector determined by the ANN, providing equivalent exploration in each output dimension. In practice, this same functionality is achieved by adding identical noise along each synapse kj between unit k in the penultimate layer and unit j in the output layer, to make the equations (and the code) more uniform between the network-, unit-, and synapse-based approaches.

For unit-based approaches (REINFORCE U and SURL), zero-mean Gaussian noise $\epsilon_j(\tau)$ is sampled independently for each unit j in the output layer, rather than once for the entire network.[4] These independent noise samples, then, can be

[2]While we here only consider a single hidden layer and backpropagating weight updates, S*RL (like REINFORCE SUN) is compatible with ANNs of arbitrary depths and architectures, and any form of gradient descent to update those layers.

[3]Because of immediate reward setup, the term τ is interchangeably used for both time step and trial.

[4]REINFORCE U is identical to REINFORCE with Gaussian units (Williams 1992), although we use different terminology and notation to ease internal comparisons.

thought of as being added to each output unit independently and provide independent exploration for each dimension of the action space. Again, however, in practice this same functionality is achieved by adding identical noise $\epsilon_j(\tau)$ along each synapse connecting to unit j in the output layer, to make the equations (and the code) more uniform.

For synapse-based approaches (that is, REINFORCE S and SSRL), zero-mean Gaussian noise $\epsilon_{kj}(\tau)$ is sampled independently along each synapse kj between unit k in the penultimate layer and unit j in the output layer.

In all cases, the noise is added to the weight value $w_{kj}(\tau)$ used in the calculation of the net value for unit j, as

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

where K is the total number of inputs from the penultimate layer, $y_k(\tau)$ is the output of unit k in that layer, and ϵ_* indicates ϵ_N, ϵ_j, or ϵ_{kj}, respectively, for the network-, unit-, or synapse-based approach. The output $o_j(\tau)$ of each unit in the output layer is computed as the logistic function of $net_j(\tau)$.

Weight Updates

To calculate weight updates, both REINFORCE SUN and S*RL use reward $r(\tau)$ and expected reward $\hat{r}(\tau)$.[5]

For REINFORCE SUN, each weight is updated using

$$w_{kj}(\tau + 1) = w_{kj}(\tau) + \eta_w(r(\tau) - \hat{r}(\tau))\epsilon_*(\tau), \qquad (2)$$

where η_w is the weights' learning rate.

For S*RL, the corresponding weight update equation is

$$w_{kj}(\tau + 1) = w_{kj}(\tau) + \eta_w(r(\tau) - \hat{r}(\tau))y_k(\tau)\epsilon_*(\tau), \qquad (3)$$

where the additional term $y_k(\tau)$ for the output from unit k is included to account for the influence of that output on the output for which the reward was received.

Exploration Parameter Updates

For REINFORCE SUN, σ_*, which determines the standard deviation of the noise Gaussian, is updated[6] using

$$\sigma_*(\tau+1) = \sigma_*(\tau) + \eta_\sigma(r(\tau) - \hat{r}(\tau))\frac{\epsilon_*(\tau)^2 - \sigma_*(\tau)^2}{\sigma_*(\tau)}, \qquad (4)$$

whereas, for S*RL, the corresponding σ_* update equation is

$$\sigma_*(\tau + 1) = \sigma_*(\tau) + \\ \eta_\sigma(r(\tau) - \hat{r}(\tau))y^*_k(\tau)(|\epsilon_*(\tau)| - c_\sigma\sigma_*(\tau)), \qquad (5)$$

where constant c_σ inclines the algorithm toward exploration or exploitation (see discussion).

Again the S*RL equation includes a term to account for the influence of the output of units in the penultimate layer on the output at the final layer, whereas the REINFORCE SUN equation does not. However, in this case the term is $y^*_k(\tau)$ to represent $y_k(\tau)$ for SSRL or $\overline{y_K}$, which is the average of all K outputs from the previous layer, for SURL and SNRL. In addition, the 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.

In all cases, σ_* updates are subject to $\sigma_*(\tau + 1) \geq 0$. That is, if Equation 4 or 5 would result in a negative value for $\sigma_*(\tau + 1)$, $\sigma_*(\tau + 1)$ is set equal to 0 (Williams 1992).

[5]Here, $\hat{r}(\tau) = d\,r(\tau - 1) + (1 - d)\,\hat{r}(\tau - 1)$, where d is a discount factor (Williams 1992). Both REINFORCE SUN and S*RL are compatible with any calculation of expected reward.

[6]This assumes that σ_* is non-zero; otherwise, σ_* remains zero.

Experimental Setup

We performed experiments using multiple data sets based on inverse kinematics of simplified models of PUMA and Stanford robotic arms with all values for all data sets scaled to be in $[0,\ 1]$. In each data set, input-output vector pairs are generated by randomly sampling joint values from a uniform distribution, generating location values based on a forward kinematics model, then using location values as inputs and joint values as target values. The kinematics 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. 50 such data sets are used for each experimental condition to collect statistically meaningful results.

For each trial τ, the ANN is presented with the input vector and a scalar reward value $r(\tau)$ based on the similarity between the agent's action vector and a target output vector from the training data on that trial, calculated using

$$r(\tau) = 1 - \frac{\sum_{j=1}^{J}|t_j - o_j|}{J}, \qquad (6)$$

where J is the number of output units, t_j is the target for unit j, and o_j is the corresponding action. The range of possible input and target values and the logistic activation functions of the output units, gives reward values in [0, 1].

The 6$\times$6 inverse kinematics problem here is to determine the values of the waist, shoulder, elbow, and three joints of a spherical wrist given the desired position in Cartesian three-space (x, y, z) and orientation given as roll, pitch, and yaw of the end effector. The 3$\times$3 data sets use the waist, shoulder, and elbow joints with the position in three-space. The 6$\times$3 data sets use the waist, shoulder, and elbow joints with position and orientation. The difference between the arms is that all PUMA joints are revolute whereas the Stanford arm has a prismatic elbow. For each data set, 20,000 input-output vector pairs are generated. The first and second halves of each data set are generated using complementary ranges of the arm joints, causing it to reach to very different positions during the first and second halves of each run, resulting in the desired dynamic environment.[7]

Each network has a $+1$ bias input in addition to the kinematics inputs and has hidden units equal to the number of the inputs plus one. The initial weights for each synapse are in $[-2, +2]$, $\lambda = 2$ for the logistic functions, all learning rates and reward discount factors are set to 0.5 (a moderate value), and σ_* is initialized uniformly randomly in $[0,\ 1]$.

[7]This setup is similar to classic continuous RL tasks (Ritter and Schulten 1987) while providing dimensionality comparable to that of popular RL tasks, such as the robotics/control tasks from Gym (https://gym.openai.com/), yet are scalable and sharply dynamic.

Results

Table 1 provides results for SSRL and REINFORCE S (the best performing variant in each group): average of average rewards $\overline{\overline{r}}$ and average of standard deviation of reward $\overline{\sigma_r}$ on all data sets over each entire run, last 1,000 trials before change, and last 1,000 trials in run. Underline indicates highest $\overline{\overline{r}}$ and lowest $\overline{\sigma_r}$. P: PUMA, S: Stanford.

Figure 1 shows average reward collected for all variants of both S*RL and REINFORCE SUN at each trial across all runs for the Stanford arm. (Similar PUMA results are not shown to conserve space.) The differences between these curves are statistically significant (randomized ANOVA, $p < 0.001$ for both algorithm and interaction for all experiments and all pairwise comparisons) (Piater et al. 1998).

Figure 2 shows box and whisker plots of reward for SSRL and REINFORCE S across all runs. Here, within each individual sub-figure, the two boxes on the left represent the cumulative reward collected during each run, the two in the middle represent the cumulative reward in the last 1,000 trials before change (when the algorithms have had time to converge), while the two on the right represent the cumulative reward received in the last 1,000 trials of the run.

These results are statistically significant for all SSRL to REINFORCE S comparisons (t-test, $p < 0.001$, $d.f.$=49) except for 3×3, 6×3, and 6×6 Stanford data sets, the last 10% of the reward collected before change (t-test, p = 0.5273, p = 0.1117, and p = 0.5394 respectively). The results are also statistically significant for all comparisons within S*RL algorithms ($p < 0.001$ for both pre and post-hoc statistics, $d.f._N$=2, $d.f._D$=98) (Quade 1979) except the following: For the 3×3 PUMA data set, the post-hoc comparison for the last 1,000 trials before change, SSRL is not significantly different from SURL (p=0.29) and after change, SURL is not significantly different from SNRL (p=0.036). Similarly, for the 6×6 Stanford data set, the post-hoc comparison for the last 1,000 trials before change, SSRL is not significantly different from SURL (p=0.26) and after change, SURL is not significantly different from SNRL (p=0.17). Further, the post-hoc comparisons for the last 1,000 trials before change shows that for 3×3 Stanford dataset SSRL is significantly different from SURL (p=0.00407), however for 6×3 PUMA, 6×3 Stanford, and 6×6 PUMA datasets, SSRL is not significantly different from SURL (p=0.059, 0.06, and 0.076 respectively).

The few results that are not statistically significant are a result of two distinct phenomena. The lone comparison where SURL results are not significantly different from those of SNRL in the last 10% of the trials after the change, are a result of SNRL achieving deceptively high reward values after the change, because it never learned well before the change, and thus gained an accidental advantage in the second half when both SSRL and SURL performance dropped well below that of SNRL because they had converged to good values prior to the change. The comparisons where SSRL results are not significantly different from their SURL and REINFORCE S counterparts in the last 10% of the rewards collected before the change are a result of all algorithms achieving near-optimal performance at that point.[8]

Table 1: Results for SSRL and REINFORCE S (in percent).

Data set	Entire run				1k trials pre change				Last 1k trials			
	SSRL		R-S		SSRL		R-S		SSRL		R-S	
	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$	$\overline{\overline{r}}$	$\overline{\sigma_r}$
3×3 P	$\underline{86.6}$	$\underline{1.7}$	80.7	4.5	93.5	$\underline{0.1}$	$\underline{93.6}$	$\underline{0.1}$	$\underline{93.3}$	$\underline{1.1}$	75.1	10.9
3×3 S	$\underline{85.6}$	$\underline{2.1}$	75.4	3.7	$\underline{91.5}$	$\underline{0.1}$	91.4	0.7	$\underline{90.4}$	$\underline{3.0}$	63.1	9.5
6×3 P	$\underline{89.7}$	$\underline{1.5}$	84.5	3.1	93.6	$\underline{0.1}$	$\underline{93.7}$	$\underline{0.1}$	$\underline{92.6}$	$\underline{2.2}$	79.0	8.0
6×3 S	$\underline{85.6}$	$\underline{2.2}$	73.4	2.6	$\underline{91.5}$	$\underline{0.1}$	91.3	1.0	$\underline{89.8}$	$\underline{3.4}$	57.5	6.1
6×6 P	$\underline{90.0}$	$\underline{0.7}$	86.9	1.7	93.4	$\underline{0.1}$	$\underline{93.7}$	$\underline{0.1}$	$\underline{93.0}$	$\underline{0.6}$	84.8	4.6
6×6 S	$\underline{87.2}$	$\underline{1.2}$	82.5	2.1	$\underline{92.4}$	$\underline{0.1}$	$\underline{92.4}$	0.4	$\underline{91.1}$	$\underline{1.6}$	77.0	5.3

Discussion

The results in aggregate overwhelmingly support both hypotheses: Each algorithm using the SSRL equations greatly outperforms each corresponding algorithm using the REINFORCE Gaussian equations in dynamic environments, and SSRL greatly outperforms SURL, which in turn greatly outperforms SNRL, in both static and dynamic environments, at least in the long term. SSRL also shows less variability than REINFORCE S in the average cumulative reward collected, making SSRL by far the most consistent in performance.

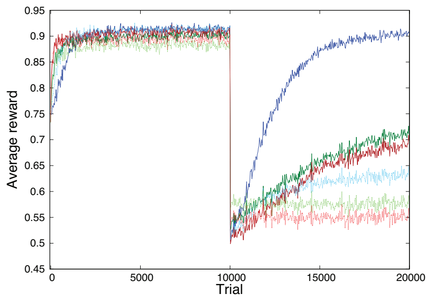
Still, stochastic units appear to learn fastest initially on many data sets and stochastic networks show faster learning initially on a few data sets and actually outperform stochastic units for the 3×3 Stanford arm in the first half of the run.

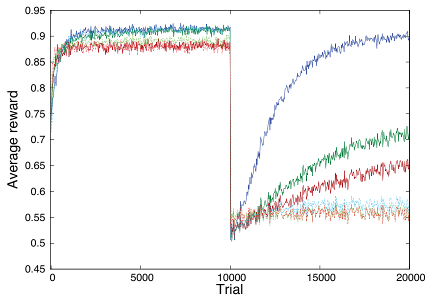
The reason that each S*RL algorithm outperforms its corresponding REINFORCE SUN algorithm clearly lies in the update equations used by each algorithm. In all other respects, the algorithms as tested are identical. However, which differences within those equations deserve credit is unclear.

One possibility is that the credit should go to the additional y terms intended to account for the influence of the output of the units from the previous layer on the output for which the reward was received. For S*RL weight updates and for SSRL σ updates, this term is y_k, which is simply the output from unit k in the previous layer. For the σ updates for both SURL and SNRL, this term is $\overline{y_K}$, which is the average of all K outputs from the previous layer. Both y_k and $\overline{y_K}$ scale the difference values calculated on the basis that the outputs from the previous layer affect the outputs at the final layer. In addition, because y_k treats the outputs from the units in the previous layer independently, this term may help with the structural credit assignment problem.

Another possibility is that the calculations of the difference between the sampled noise ϵ_* and the standard deviation σ_* in the σ_* update equation for S*RL (Eq. 5) deserves the credit. In the case of REINFORCE (and, therefore, REINFORCE SUN), the corresponding calculation comes from the partial derivative with respect to σ of the natural log of the normal distribution, with a factor of σ^2 thrown in to obtain a “reasonable algorithm” (Williams 1992, p. 10). In contrast, for SSRL (and, therefore, S*RL) this calculation comes from a straightforward intuition of how such an algorithm should perform: “If the agent is performing better than expected, the

[8]Note that performance and statistical results for SURL vs. REINFORCE U and SNRL vs. REINFORCE N (not shown for brevity) largely follow similar trends. Source code and all data are available at https://doi.org/10.5281/zenodo.2629416.


(a) 3×3 Stanford Arm Data Set.


(b) 6×3 Stanford Arm Data Set.

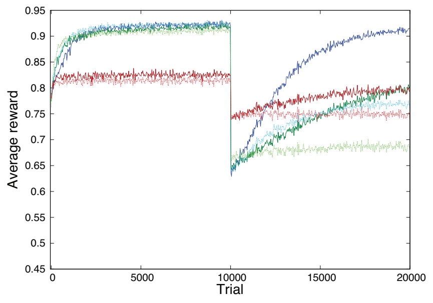
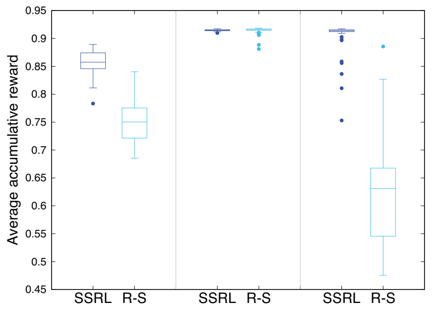
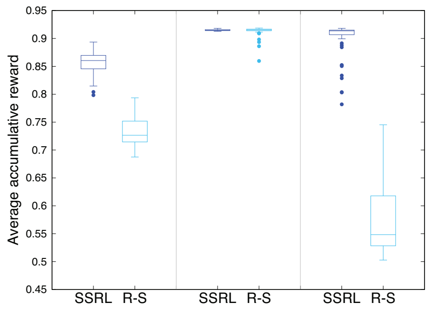

(c) 6×6 Stanford Arm Data Set.

Figure 1: S*RL and REINFORCE SUN average reward collected per trial across 50 runs by each algorithm. Dark blue: SSRL. Light blue: REINFORCE S. Dark green: SURL. Light green: REINFORCE U. Dark red: SNRL. Light red: REINFORCE N.


(a) 3×3 Stanford Arm Data Set.


(b) 6×3 Stanford Arm Data Set.

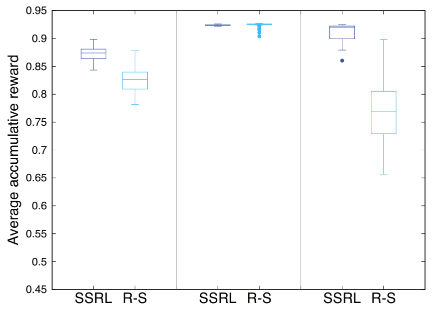

(c) 6×6 Stanford Arm Data Set.

Figure 2: SSRL and REINFORCE S (abbreviated R-S) average cumulative reward across 50 runs. Box and whisker plots with outliers. First third in each plot shows average cumulative reward over entire run while second third shows same for last 1,000 trials before change and last third shows last 1,000 trials before end of run.

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" (Shah and Hougen 2017, p. 4). Taking this notion at face value, the SSRL equation simply takes the difference between the absolute value of the particular sampled noise ϵ_* and the standard deviation of the noise distribution itself σ_* times the constant c_σ. This constant can be used to bias the algorithm toward exploration or exploration. Here, c_σ is set to 0.6745 to leave the algorithm unbiased.[9]

Of course, it could be that the credit should go to some combination of these factors, or it might even be that some of these factors are contributing to the superior performance of S*RL while others are actually detracting. This credit assignment problem is left as future work.

Conclusions

This paper introduces S*RL, a superclass that generalizes SSRL to allow for stochasticity at the level of the synapse, the unit, or the network and empirically compares S*RL to REINFORCE SUN, itself a generalization of the venerable REINFORCE class of RL algorithms. This study clearly demonstrates the superiority of each of S*RL's options as compared to each corresponding option in REINFORCE SUN in dynamic environments with continuous states and actions, and makes it clear that the equations governing S*RL are responsible for its greater success in these environments. These results strongly suggest that practitioners should prefer S*RL to REINFORCE SUN in dynamic environments.

This study also adds support for the general hypothesis that finer-grained stochasticity outperforms coarser-grained stochasticity, at least for a range of interesting continuous-state-continuous-action problems, as well as the more specific hypothesis that stochastic synapses outperform both stochastic units and networks, at least in the long term. As one would expect, however, stochastic synapses have to explore at a more granular level and thus take longer to learn initially due to tuning substantially more parameters compared to stochastic units and networks. This suggests that a hybrid approach might combine the best of all variants; that is, using stochastic units and/or networks initially might give faster early learning, and switching to stochastic synapses later might achieve optimal performance in a long run.

Putting these ideas together, this study strongly suggests that practitioners should prefer SSRL to SURL, SNRL, or any version of REINFORCE SUN (including the original) in dynamic continuous state-continuous action environments,

[9]In a Gaussian distribution, 50% of samples are within 0.6745σ of the mean.

and that researchers should consider SSRL to be the new benchmark algorithm in these environments and should look to incorporate SSRL in place of REINFORCE as the core element in sophisticated, state-of-the-art RL systems.

Nevertheless, the results are far less clear for other environments. In particular, for static environments it appears that SSRL learns more slowly than many of the options considered here, as evidenced by its noticeably shallower ascent at the start of many of the graphs. These results suggest that for static environments in which early performance is crucial, one should prefer REINFORCE S or perhaps even the original REINFORCE (which corresponds to REINFORCE U) if long-term performance should be sacrificed for short-term returns. However, if the environment is not known to be static, the brief underperformance of SSRL is outweighed by its overwhelmingly superior performance in the dynamic environments. Still, the hybrid approach of starting stochasticity at the level of the network or the unit and moving it to the level of the synapse as learning progresses seems even more important for S*RL than for REINFORCE SUN.

Future Work

While SSRL outperforms alternatives in dynamic environments with immediate rewards and continuous states and actions, other environments such as those with discrete states and/or those with delayed rewards should be considered. SSRL has shown promise in such environments (Shah and Hougen 2017) but direct comparisons are absent.

As noted in the conclusions, it is unclear which differences between the S*RL and REINFORCE SUN equations deserve credit for S*RL's greater performance. Investigating this credit assignment problem is important future work.

As already suggested, a hybrid approach with potentially better long-term performance (the strength of stochastic synapses) yet faster initial learning (the strength of both stochastic units and networks), should be studied.

Further, deep RL studies with very high dimensional state and action spaces using stochastic synapses can potentially improve on existing results. Similarly, a future study on comparing deterministic policy gradient algorithms to SSRL can shed more light on whether our proposed method performs better in spite of stochastic policy gradient algorithms requiring more samples as claimed (Silver et al. 2014).

References

Amravati, A.; Nasir, S. B.; Thangadurai, S.; Yoon, I.; and Raychowdhury, A. 2018. A 55nm time-domain mixed-signal neuromorphic accelerator with stochastic synapses and embedded reinforcement learning for autonomous micro-robots. In *2018 IEEE International Solid-State Circuits Conference (ISSCC)*, 124–126.

Deisenroth, M. P.; Neumann, G.; and Peters, J. 2011. A survey on policy search for robotics. *Foundations and Trends in Robotics* 2(1-2):1–142.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338.

Engelbrecht, A. P. 2007. *Computational Intelligence: An Introduction*. Wiley, second edition.

Heess, N.; Wayne, G.; Silver, D.; Lillicrap, T.; Erez, T.; and Tassa, Y. 2015. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc. 2944–2952.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2019. Continuous control with deep reinforcement learning. *arXiv:1509.02971v6*.

Ma, X., and Likharev, K. K. 2006. Global reinforcement learning in neural networks with stochastic synapses. In *Proceedings of the 2006 IEEE/INNS International Joint Conference on Neural Networks*, 47–53.

Mao, H.; Venkatakrishnan, S. B.; Schwarzkopf, M.; and Alizadeh, M. 2019. Variance reduction for reinforcement learning in input-driven environments. *arXiv:1807.02264v3*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Piater, J. H.; Cohen, P. R.; Zhang, X.; and Atighetchi, M. 1998. A randomized ANOVA procedure for comparing performance curves. In *Proceedings of the International Conference on Machine Learning*, volume 98, 430–438.

Quade, D. 1979. Using weighted rankings in the analysis of complete blocks with additive block effects. *Journal of the American Statistical Association* 74(367):680–683.

Ritter, H., and Schulten, K. 1987. Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In *Neural Computers*, volume F41. Springer. 393–406.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2018. High-dimensional continuous control using generalized advantage estimation. *arXiv:1506.02438v6 [cs]*.

Shah, S. N. H., and Hougen, D. F. 2017. Stochastic synapse reinforcement learning (SSRL). In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1–8.

Shah, S. N. H., and Hougen, D. F. 2019. Rethinking stochasticity in neural networks for reinforcement learning with continuous actions. In *IEEE Symposium Series on Computational Intelligence*, 9 pages.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, 387–395.

Sutton, R. S., and Barto, A. G. 1981. Toward a modern theory of adaptive networks: Expectation and prediction. *Psychological Review* 88(2):135–170.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, King's College.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8(3-4):229–256.