Evaluation of Additive Noise Methods to Aid Agent Exploration in Deep Policy Gradient Methods

Nathaniel Hanson, Eric Grimaldi
https://github.com/nhanson2/ReinforcementRobotExploration
CS5180 Reinforcement Learning
December 15, 2021

1 Abstract

A hallmark of Reinforcement Learning (RL) is the question: explore or exploit? As agents are placed in an unfamiliar environment and must learn policies for how to perform optimally, one must be careful to prevent an agent for becoming self-assured in a solution, when it has only explored a small sliver of the total state-action space. In classical RL problems, such as grid world or Black Jack, which contain discrete state and action spaces, exploration can be managed by classical methods such as ϵ-greedy policies or Upper Confidence Bound (UCB) to algorithmically encourage non-optimal actions in the hope that one of these actions might yield a better outcome than the current policy. However, these type of problems represent a sliver of real world problems. For instance, robotic manipulation contains both a large observation state space of object positions and manipulator positions in addition to a continuous action space to command joint positions and velocities.

Using classical learning and exploration methods, discovery of an optimal policy is doomed to fail. Policy gradient methods have been used to converge to policies for continuous state spaces, but the implications of exploration methods within this discipline is not well understood. In our project, we seek to understand current exploration methods used in policy gradient learning and apply them to solving continuous observation-action problems. Namely, we implement Gaussian White Noise, ϵ-Greedy, ϵ-Greedy Decay, Ornstein-Uhlenbeck Process (OU) [1], Upper Confidence Bound (UCB), Parameter Space Exploration, and a Counter-Based Hash. We compare these methods against learning with no exploration as a baseline.

We hypothesized the parameter space exploration, discussed in detail in Section 3, would lead to the fastest learning and converge to a better solution than the oft-employed additive Gaussian action space noise. For use with a Deep Deterministic Policy Gradient, these methods show considerable variance. When Hindsight Experience Replay (HER) is added to improve learning in high dimensional state spaces with sparse rewards, the methods offer nearly no significant improvement over a policy with programmed exploration. This surprising result is analyzed in Section 8.

2 Project Overview

Policy gradients methods aim to find an optimal policy by optimizing the policy instead of $Q(s,a)$. The policy $\pi_\theta(a)$ is optimized by finding parameters θ in order to achieve the maximum expected reward. By descending the gradient of the policy, we are able to arrive at a set of weights which minimize the expected loss, as is the case with many gradient-descent optimization algorithms. Policy gradients are generally useful in RL, but the subset of Deep Deterministic Policy Gradients (DDPG) has proven wildly successful for continuous action spaces [2]. DDPG uses an architecture that maintains two neural networks as listed:

1. Actor (Target Policy) $\pi(S) \longrightarrow A$
2. Critic (Value Approximator) $Q(S, A) \longrightarrow \mathbb{R}$

Similar to how Deep Q-Learning is executed, we utilize a series of mini-batches consisting of sampled transitions from past experiences $[state,\ action,\ reward,\ state',\ done]$ in a replay buffer. In DDPG, the loss functions is given by:

$$y(r, s', d) = r + \gamma(1-d)Q_\theta(s', \pi_\theta(s'))$$

And for a mini-batch consisting of B experiences, the critic and actor networks, respectively, are updated by calculating:

$$\nabla \frac{1}{|B|} \sum_{s,a,r,s',d \in B} (Q_\theta(s,a) - y(r,s',d))^2 \text{ (A Quadratic loss function)}$$

$$\nabla \frac{1}{|B|} \sum_{s,a,r,s',d \in B} Q_\theta(s, \pi_\theta(s))$$

As the algorithm is learning through playing multiple episodes, noise is injected into the system to induce exploration. Classically, this is implemented by adding zero-mean Gaussian white noise $\mathcal{N}(0,1)$ to the target policy action selection. This noise is commensurate with a stochastic transition function, where the action chosen is not always the action applied to the environment.

DDPG has been show to perform well on a number of classic video games and control problems. However, it struggles when the reward signal is sparse. A team of researchers at OpenAI proposed an improvement to DDPG called Hindsight Experience Replay (HER) to handle sparse reward environments [3]. In RL, clever reward engineering can be used to help guide agents to an optimal policy. However, reward shaping requires prior knowledge of the environment and what constitutes "good" or "bad" for intermediary actions. A simpler solution which more closely mirrors real-world scenarios is to only have reward at the end of the scenario ending in either success, or failure. This poses a problem for normal RL strategies, since generally duration of the episode adds some information about the relative goodness of the trial. To handle the sparse reward problem, HER attempts to learn something from every trial. HER considers trajectories: $[(S_0, G, a_0, r_0, S_1), (S_1, G, a_1, r_1, S_2), ..., (S_n, G, a_n, r_n, S')]$. Even if the goal G is not achieved for the trajectory, HER contextualizes the trajectory as

$$[(S_0, S', a_0, r_0, S_1), (S_1, S', a_1, r_1, S_2), ..., (S_n, S', a_n, r_n, S')]$$

This approach treats the final state as if it had been the goal all along, and the agent now has experience, leveraging its action space to achieve it - valuable information we can exploit in the learning process. HER requires inclusion of a goal in the environmental model in order to succeed.

3 Prior Work in Exploration

The original implementation of DDPG suggests using a Gaussian process or Ornstein-Uhlenbeck (OU) based process to induce exploration. OU noise is a temporally correlated Markov process where the noise can be thought of as having "momentum", since it is added to both it previous state before adding to the current action space. Temporally, the net effect of such actions trends averages to 0, but it can lead to successive actions diverging from the current policy, as show in Fig. 3.1. Also thought of as Brownian Motion, these processes are random walks and diverge further from the mean than white noise tends to.

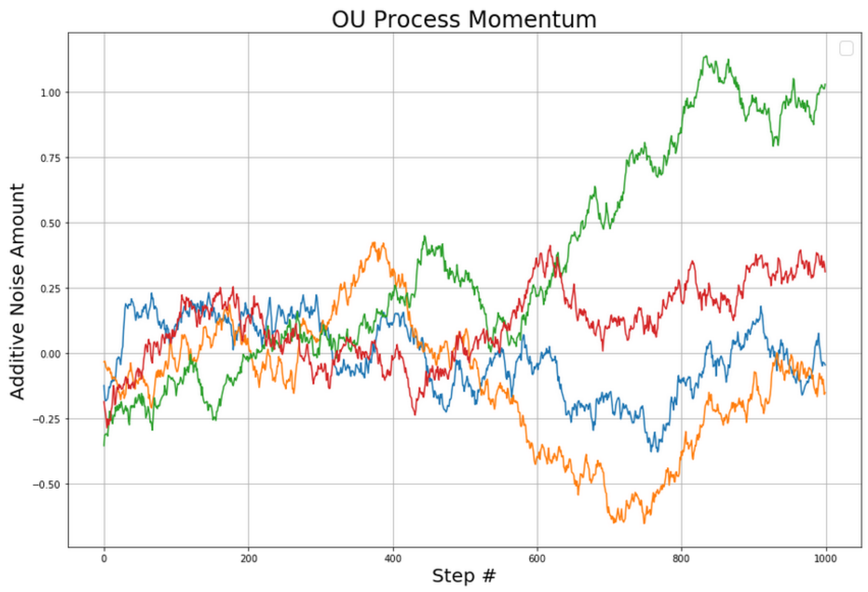

Figure 3.1: Four dimensional OU walk process over 1000 time steps

The only other major work that has been put forth in understanding alternate strategies in continuous state-action space is a 2017 paper by Plappert et al [4]. Instead of injecting noise directly to the applied actions, noise is applied to the parameters of the policy: $\tilde{\pi} \triangleq \pi_\theta \mid \theta \leftarrow \theta + \mathcal{N}(0,1)$. In doing so, the agent learns to in spite of a noisy parameterization. As with the additive action noise, this process requires the actions to be clipped to appropriate bounds.

4 Technical Statements & Hypotheses

We hypothesize the parameter space exploration, discussed in detail in Section 3 will drastically outperform the other methods of inducing exploration. We especially expect to see these algorithms outperform naive strategies such as e-greedy action space sampling. We expect similar results in both DDPG, and in DDPG + HER, although we expect faster convergence to a stable policy when using HER.

5 Environments

For our project, we utilized two environments included as part of the OpenAI Gym ecosystem. We choose this platform since it provides an abstracted platform to create agents and interact with environments in a repeatable, generalized way. In previous assignments we have used OpenAI environments with discrete or continuous observation spaces, but with discrete action spaces. However, several of these environments also offer a more challenging continuous state space. We first consider **Mountain Car** as a simple physics-based simulation. The simple task for environment is to move an underpowered car from the trough between two hills to the peak of the rightmost one. Only by building momentum is the car able to achieve a great enough velocity to escape. Specific environment structure follows:

$$\text{Observation Space: } s = \begin{bmatrix} x \in [-1.2,\ 0.6] \\ \dot{x} \in [-0.07,\ 0.07] \end{bmatrix} \text{ Position of the car}$$

$$\text{Action Space: } a \in [-1.0,\ 1.0] \text{ Power coefficient}$$

$$\text{Reward Function: } R(s,a) = \begin{cases} 100 & \text{Agent at top of right mountain} \\ -(a^2 \times 0.1) & \text{Otherwise} \end{cases}$$

An additional consideration: the simulator will time out if the agent takes more than 200 steps to complete the task.

Secondly, we utilized a much more complicated robotics environment. We use a Fetch mobile manipulator in simulation with the Multi-Joint dynamics with Contact (Mujoco) physics engine. The **Fetch slide** environment contains a single robot stationary next to a table. The table contains a single puck with a low coefficient of friction. Using its 7 degrees of freedom (DoF) manipulator, the robot must impact the put with its end effector to slide it to a goal position. This simulator assumes perfect state knowledge; puck position is known in terms of global environment coordinates. No advanced image processing or visual information is input into a learning algorithm. To simplify the environment, the end effector position is locked in a set orientation by only permitting commands to the first 4 joints.

$$\text{Observation Space: } s = \begin{bmatrix} \mathbb{R}^7\text{, Gripper absolute position} \\ \mathbb{R}^3\text{, Gripper linear velocity} \\ \mathbb{R}\text{, Distance between fingers} \\ \mathbb{R}\text{, Velocity between fingers} \\ \mathbb{R}^3\text{, Object position} \\ \mathbb{R}^3\text{, Object rotation} \\ \mathbb{R}^3\text{, Object position relative to gripper} \\ \mathbb{R}^3\text{, Object linear velocity} \\ \mathbb{R}^3\text{, Object angular velocity} \end{bmatrix}$$

$$\text{Action Space: } a = \begin{bmatrix} \mathbb{R}\text{, Desired relative gripper position} \\ \mathbb{R}\text{, Desired distance between fingers} \end{bmatrix}$$

$$\text{Reward Function: } R(s,a) = \begin{cases} 0 & \text{Goal position not achieved} \\ 1 & \text{Goal position achieved} \end{cases}$$

This reward structure for the Fetch robot is a *sparse* rewards environment. Since the only reward achieved is upon success, the agent has little information to learn from in unsuccessful trials. We will address strategies to overcome this challenge in Section 6.

6 Methods & Algorithms

1. No Exploration:
   This trivial case is provided primarily as a point of comparison. No exploration is not particularly good as an exploration method, however the inherent randomness of DDPG due to random initializations and stochastic gradient descent result in a small but incomplete amount of inherent exploration. If nothing else, the inclusion of this trivial case will help highlight exploration methods which are counter productive.

2. Gaussian White Noise:
   Every time a new action is taken, white noise is added to the action space, sampled from a Gaussian distribution $\mathcal{N}(0,1)$, where each action space dimension is a unique sampled quantity. I.e., the same value is not added to each dimension. Values are clipped to the min-max range of the action space.

3. Ornstein-Uhlenbeck Process:
   This method is nearly identical to the above Gaussian White Noise method, however the additive Noise term is now generated by an OU Process; this is defined by the stochastic differential equation

$$dx_t = -\theta x_t dt + W_t$$

   where θ and σ are parameters greater than 0, x_t is the OU process, and W_t is the Wiener Process (essentially a Gaussian random walk). In essence, the OU process is a random walk with a momentum term.

4. ϵ-Greedy:
   Actions are chosen greedily, except some percentage ϵ of the time where actions are chosen randomly. Since we are working with continuous action spaces, actions are chose uniformly randomly from the action space.

5. ϵ-Greedy Decay:
   This method is identical to ϵ-Greedy, except that ϵ begins rather large and decays according to a schedule over the course of training to nearly 0.

6. Counter:
   Counting state-action pairings in continuous space is a very difficult task to accomplish, especially in the high dimensional space presented by Fetch. As an intermediary solution, we look to function approximation, and tile [5] the environment in 1000 tiles, and hash the environmental state to generate a lookup table for overlapping state spaces. Since reverse sampling from these tiles would prove exceptionally difficult in practice, we use the tile counts and total actions taken to inform the scope of additive noise. If an action has been taken several times, we decrease the total amount of noise allowed by $a \leftarrow a + \mathcal{N}(0, \frac{\beta}{\sqrt{||Tiles(s,a)||}})$. β is a tuned hyperparameter, set as 0.5 for the experiments.

7. Upper Confidence Bound (UCB) [6]:
   Utilizing our hashing developed for the above Counter method, we look to implement UCB exploration. In this method, we calculate a measure of certainty in out estimates of value. As the ratio of samples of a state to all samples dwindles, our certainty in the estimate decreases. We select actions greedily, but with a term added to the value of the action inversely proportional to our certainty. In essence actions are chosen according to $a_t = a_t + c\sqrt{\frac{\log t}{N_t(a)}}$ where c is a parameter greater than 0, t is the current step number, and $N_t(a)$ is the current hashed count for action a.

8. Parameter Space Noise:
   To implement parameter space noise, we add additional layers between each of the fully-connected layers in the actor-critic networks. After forward passing the input observation tensor through the layer, we add Gaussian white noise to each of the weights in the network, perturbing the input before passing to the next layer. This layer is only active on the forward pass, so it is not optimized on the backward pass.

7 Model

Our base actor and critic networks are three layer fully-connected neural networks taking in a 1×28 or 1×3 input observation vector, and generating a 1×4 or 1×1 action in continuous space, depending on the environment. Our hidden layer size is 256 consistently throughout the layers. To create a non-linearity, we use ReLU as our activation after each fully connected layer. We use the Adam optimizer [7] to schedule a learning rate, starting at $lr = 0.001$. We train the networks on a GeForce RTX 3080 GPU. For Mountain Car, training takes approximately 7 minutes to converge. Fetch-Slide takes 1.25 hours to converge after 200 epochs.

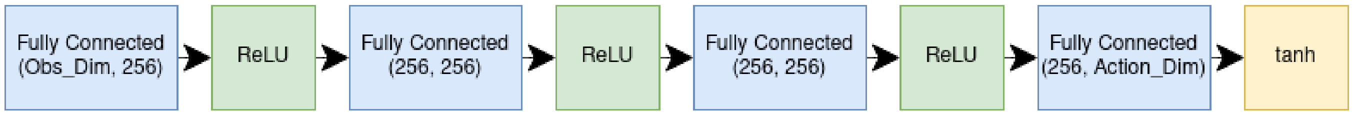
Fully Connected (Obs_Dim, 256)
ReLU
Fully Connected (256, 256)
ReLU
Fully Connected (256, 256)
ReLU
Fully Connected (256, Action_Dim)
tanh

Figure 7.1: Actor network for estimating policy

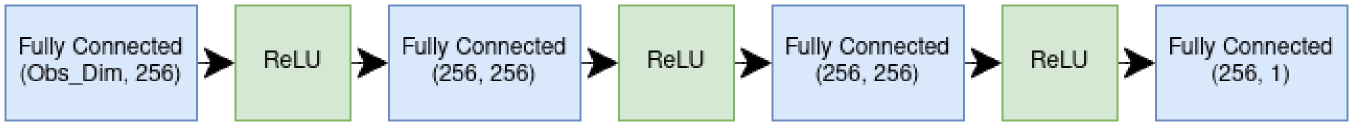
Fully Connected (Obs_Dim, 256)
ReLU
Fully Connected (256, 256)
ReLU
Fully Connected (256, 256)
ReLU
Fully Connected (256, 1)

Figure 7.2: Critic network for estimating Q-Value function

8 Empirical Results

After observing stability in the model and hyperparameters, each method was run 5 times with random initialization. For Mountain Car, we plot the total discounted reward per episode. For Fetch, we plot the number of successful trials for each episode. Confidence bounds are calculated as $\mu_{res} \pm 1.96 \times \sigma_{res}$.

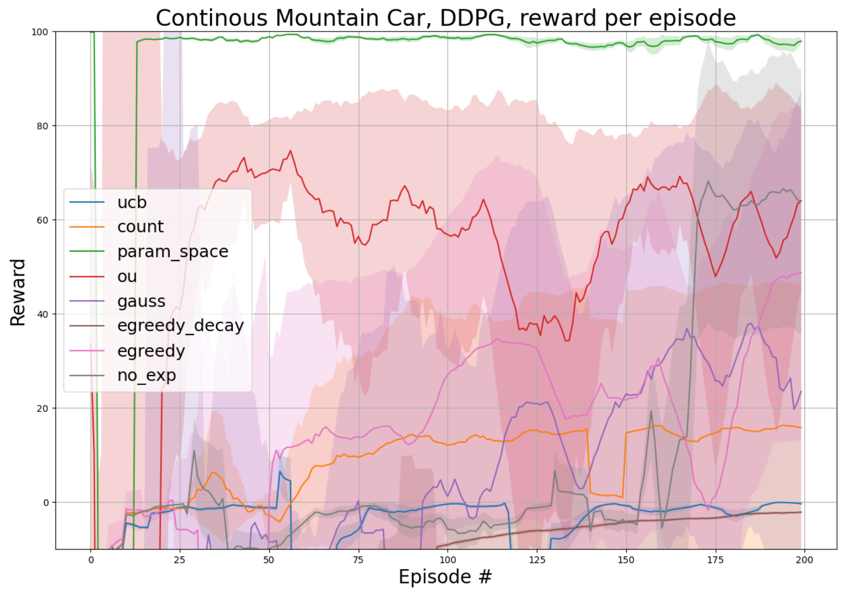

Figure 8.1: Averaged results for DDPG and a variety of exploration methods on the Continuous Mountain Car problem.

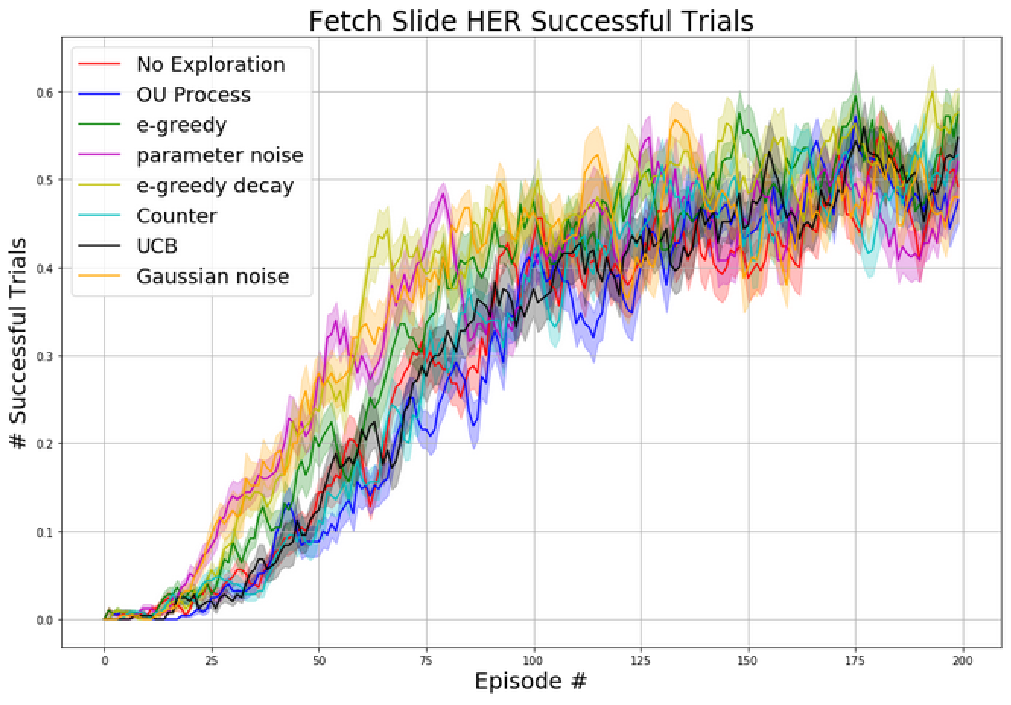

Figure 8.2: Averaged results for DDPG + HER and a variety of exploration methods on the Fetch slide problem.

9 Analysis

9.1 Mountain Car

In our Mountain Car results, one thing is immediately: several of the methods investigated are highly variant; the limit of 5 trials due to time considerations is simply an unfortunate limitation of our work. One should also note that despite this, the parameter space exploration method achieve an incredibly high reward incredibly quickly and incredibly consistently. OU process exploration also performs quite well, though with higher variance. Some methods such as ϵ-greedy, hash count, and Gaussian exploration achieve middling performance in the episodes allotted; it appears they need more episodes to converge. Meanwhile methods such as ϵ-greedy decay and upper confidence bound exploration struggle to "get off the ground" at all in the episodes allotted. The trivial case of no exploration method at all performed surprisingly well: it initially appears to make no progress for quite some time, but then suddenly shoots up to a reward high enough to rival OU process exploration.

9.2 Fetch Slide

The most significant result from the fetch trials is the relative similarity of the results. All trials converged to a similar success rate of approximately 55%. This success rate is deceptively low. When observing the visual results from the training, the robot nearly always manages to slide the puck to within a close distance of the goal area. However, the constrained bounding box tolerance prevents additional trials from being marked as successful. Nonetheless, the similarity of results is a testament to the power of HER to learn from failure, and experiences in general. Because the end result of the episode is always encoded as the *desired* goal of the trajectory after episode termination, the agent is able to use this experience to improve its policy. Parameter noise and ϵ-greedy decay initially outperform the other methods, likely because ϵ-greedy starts exploring early and eventually decays to follow the policy in most situations. As hypothesized, parameter noise starts out the best, but becomes indistinguishable after the 100^{th} episode. Additionally significant is the performance of the "No Exploration" strategy. It performs as well as UCB, OU Process, and Gaussian Noise simply by following the policy. By treating the high number of initial failures as valid successes, it appears the agent is able to derive sufficient weights. Although we do not observe a larger number of trials with asymptotic behavior, the original authors of the HER paper suggest 200 training epochs for the agent converge to the best learnable policy.

10 Discussion & Future Directions

HER nearly eliminates the need for exploration in DDPG. In essence, HER turns even a failure into a a valuable data point for learning. Without HER, DDPG benefits from a reasonable exploration method for the environent at hand. The original designers of DDPG suggested that simple Gaussian noise is appropriate, but more complex environments may require more complex exploration. In the mountain car case, Gaussian noise can lead to highly counter productive action selection, which can slow down training and induce higher variance. Ornstein-Uhlenbeck noise alleviates this somewhat process by adding a momentum term which is highly relevant for the mountain car environment, where manipulating the momentum of the car is a key part of the solution. Exploration through parameter space noise seems highly promising: since this method injects noise to the observations given to the agent and not the behavior of the agent, it avoids the classical problem of residual non-optimality in on-policy learning due to the necessity of exploration. No exploration at all is also surprisingly effective: it struggles to find an initial success for some time, but quickly converges after doing so. No exploration is prone to local minima, the inherent randomness of DDPG alleviates this somewhat, but it is ill advised to rely on this. The naive method of ϵ-greedy suffers severely in this context, as the uniform randomness of the exploration case is prone to obliterating any momentum the car has built up. A particularly clever ϵ schedule may alleviate this problem. Our work maintained the simple standard of a decaying schedule from very large to very small ϵ; our results highlighted that the high initial values in the schedule may be highly counter productive compared to a small static ϵ. In future of iterations of this work, we would like to see if these learning strategies translate well to the real world. For simpler RL problems, parameter space noise might enable real-time learning, as was demonstrated on the mountain car.

REFERENCES

[1] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[2] 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 preprint arXiv:1509.02971*, 2015.

[3] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5055–5065, 2017.

[4] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," *arXiv preprint arXiv:1706.01905*, 2017.

[5] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *International Symposium on Abstraction, Reformulation, and Approximation*, pp. 194–205, Springer, 2005.

[6] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *Journal of Machine Learning Research*, vol. 3, no. Nov, pp. 397–422, 2002.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.