Supplementary Material for Striving for Simplicity and Performance in Off-Policy DRL: Output Normalization and Non-Uniform Sampling

1. Hyperparameters

Table 1 shows hyperparameters used for SOP, SOP+ERE and SOP+PER. For adaptive SAC, we use our own PyTorch implementation for the comparisons. Our implementation uses the same hyperparameters as used in the original paper (Haarnoja et al., 2018). Our implementation of SOP variants and adaptive SAC share most of the code base. For TD3, our implementation uses the same hyperparamters as used in the authors' implementation, which is different from the ones in the original paper (Fujimoto et al., 2018). They claimed that the new set of hyperparamters can improve performance for TD3. We now discuss hyperparameter search for better clarity, fairness and reproducibility (Henderson et al., 2018; Duan et al., 2016; Islam et al., 2017).

For the η value in the ERE scheme, in our early experiments we tried the values (0.993, 0.994, 0.995, 0.996, 0.997, 0.998) on the Ant and found 0.995 to work well. This initial range of values was decided by computing the ERE sampling range for the oldest data. We found that for smaller values, the range would simply be too small. For the PER scheme, we did some informal preliminary search, then searched on Ant for β_1 in (0, 0.4, 0.6, 0.8), β_2 in (0, 0.4, 0.5, 0.6, 1), and learning rate in (1e-4, 2e-4, 3e-4, 5e-4, 8e-4, 1e-3), we decided to search these values because the original paper used $\beta_1 = 0.6$, $\beta_2 = 0.4$ and with reduced learning rate. For the exponential sampling scheme, we searched the λ value in (3e-7, 1e-6, 3e-6, 5e-6, 1e-5, 3e-5, 5e-5, 1e-4) in Ant, this search range was decided by plotting out the probabilities of sampling, and then pick a set of values that are not too extreme. For σ in SOP, in some of our early experiments with SAC, we accidentally found that $\sigma = 0.3$ gives good performance for SAC without entropy and with Gaussian noise. We searched values (0.27, 0.28, 0.29, 0.3). For σ values for TD3+, we searched values (0.1, 0.15, 0.2, 0.25, 0.3).

2. Entropy Value Comparison

To better understand whether the simple normalization term in SOP achieves a similar effect compared to explicitly maximizing entropy, we plot the entropy values for SOP and SAC throughout training for all environments. Figure 1 shows that the SOP and SAC policies have very similar entropy values across training, while removing the entropy term from SAC leads to a much lower entropy value. This indicates that the effect of the action normalization is very similar to maximizing entropy.

3. ERE Pseudocode

Our Streamlined Off Policy (SOP) with Emphasizing Recent Experience (ERE) algorithm is described in Algorithm 1.

4. Inverting Gradient Method

In this section we discuss the details of the Inverting Gradient method.

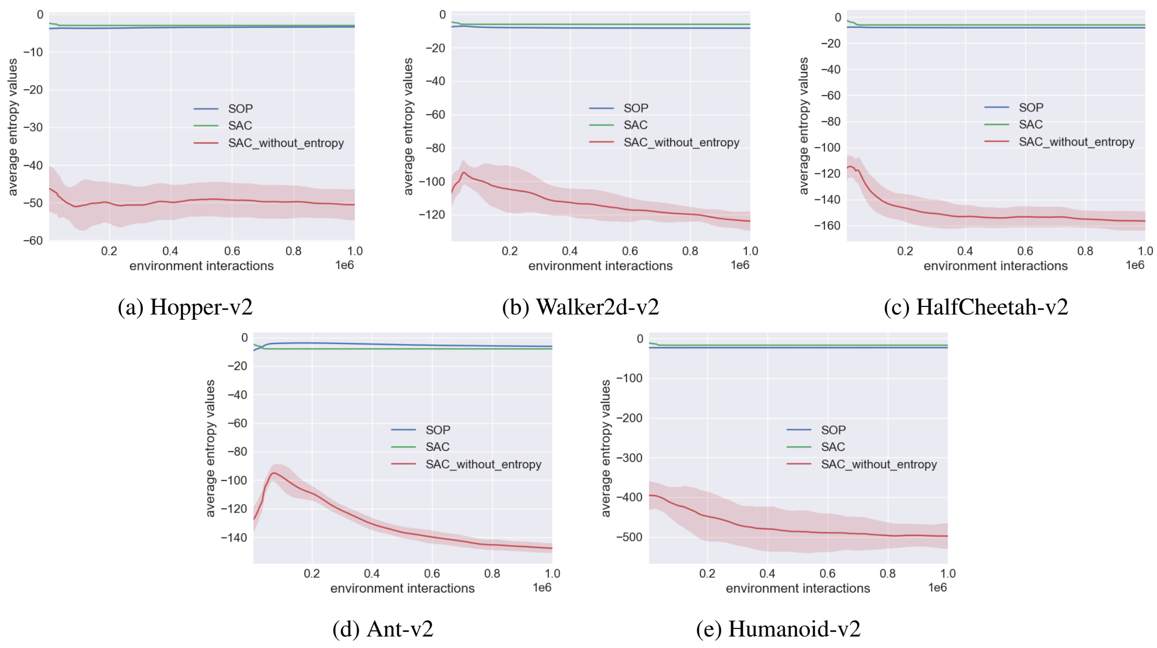
Hausknecht & Stone (2015) discussed three different methods for bounded parameter space learning: Zeroing Gradients, Squashing Gradients and Inverting Gradients, they analyzed and tested the three methods and found that Inverting Gradients method can achieve much stronger performance than the other two. In our implementation, we remove the tanh function from SOP and use Inverting Gradients instead to bound the actions. Let p indicate the output of the last layer of the policy network. During exploration p will be the mean of a normal distribution that we sample actions from, the IG approach can be summarized by the following equation (Hausknecht & Stone, 2015):

$$\nabla_p = \nabla_p \cdot \begin{cases} \frac{p_{\max}-p}{p_{\max}-p_{\min}} & \text{if } \nabla_p \text{ suggests increasing } p \\ \frac{p-p_{\min}}{p_{\max}-p_{\min}} & \text{otherwise} \end{cases} \tag{1}$$

Where ∇_p is the gradient of the policy loss w.r.t to p. During a policy network update, we first backpropagate the gradients from the outputs of the Q network to the output of the policy network for each data point in the batch, we then compute the ratio $(p_{\max} - p)/(p_{\max} - p_{\min})$ or $(p_{\max} - p)/(p_{\max} - p_{\min})$ for each p value (each action dimension), depending on the sign of the gradient. We then backpropagate from the output of the policy network to parameters of the policy network, and we modify the gradients in the policy network according to the ratios we computed. We made an efficient implementation and further

Table 1: SOP Hyperparameters

Parameter	Value
Shared	
optimizer	Adam (Kingma & Ba, 2014)
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
target smoothing coefficient (ρ)	0.005
target update interval	1
replay buffer size	10^6
number of hidden layers for all networks	2
number of hidden units per layer	256
mini-batch size	256
nonlinearity	ReLU
SAC adaptive	
entropy target	-dim($\mathcal{A}$) (e.g., 6 for HalfCheetah-v2)
SOP	
gaussian noise std $\sigma = \sigma_1 = \sigma_2$	0.29
TD3	
gaussian noise std for data collection σ	0.1 * action limit
guassian noise std for target policy smoothing $\tilde{\sigma}$	0.2
TD3+	
gaussian noise std for data collection σ	0.15
guassian noise std for target policy smoothing $\tilde{\sigma}$	0.2
ERE	
ERE initial η_0	0.995
PER	
PER β_1 (α in PER paper)	0.4
PER β_2 (β in PER paper)	0.4
EXP	
Exponential λ	$5e-06$



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2 (d) Ant-v2 (e) Humanoid-v2

Figure 1: Entropy value comparison between SOP, SAC, and SAC without entropy maximization

discuss the computation efficiency of IG in the implementation details section.

5. SOP with Other Sampling Schemes

We also investigate the effect of other interesting sampling schemes.

5.1. SOP with Prioritized Experience Replay

We also implement the proportional variant of Prioritized Experience Replay (Schaul et al., 2015) with SOP.

Since SOP has two Q-networks, we redefine the absolute TD error $|\delta|$ of a transition (s, a, r, s') to be the average absolute TD error in the Q network update:

$$|\delta| = \frac{1}{2} \sum_{l=1}^{2} |y_q(r, s') - Q_{\phi,l}(s, a)| \tag{2}$$

Within the sum, the first term $y_q(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tanh(\mu_\theta(s') + \delta))$, $\delta \sim \mathcal{N}(0, \sigma_2)$ is simply the target for the Q network, and the term $Q_{\theta,l}(s, a)$ is the current estimate of the l^{th} Q network. For the i^{th} data point, the definition of the priority value p_i is $p_i = |\delta_i| + \epsilon$. The probability of sampling a data point $P(i)$ is computed as:

$$P(i) = \frac{p_i^{\beta_1}}{\sum_j p_j^{\beta_1}} \tag{3}$$

where β_1 is a hyperparameter that controls how much the priority value affects the sampling probability, which is denoted by α in Schaul et al. (2015), but to avoid confusion with the α in SAC, we denote it as β_1. The importance sampling (IS) weight w_i for a data point is computed as:

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta_2} \tag{4}$$

where β_2 is denoted as β in Schaul et al. (2015).

Based on the SOP algorithm, we change the sampling method from uniform sampling to sampling using the probabilities $P(i)$, and for the Q updates we apply the IS weight w_i. This gives SOP with Prioritized Experience Replay (SOP+PER). We note that as compared with SOP+PER, ERE does not require a special data structure and has negligible extra cost, while PER uses a sum-tree structure with some additional computational cost. We also tried several variants of SOP+PER, but preliminary results show that it is unclear whether there is improvement in performance, so we kept the algorithm simple.

5.2. SOP with Exponential Sampling

The ERE scheme is similar to an exponential sampling scheme where we assign the probability of sampling according to the probability density function of an exponential distribution. Essentially, in such a sampling scheme, the more recent data points get exponentially more probability of being sampled compared to older data.

For the i^{th} most recent data point, the probability of sampling a data point $P(i)$ is computed as:

$$P(i) = \lambda e^{-\lambda x} \tag{5}$$

Algorithm 1 SOP with Emphasizing Recent Experience

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$ of size $N$, initial $\eta_0$, recent
   and max performance improvement $I_{recent} = I_{max} = 0$.
2: Set target parameters equal to main parameters $\phi_{\text{targ,i}} \leftarrow \phi_i$ for i = 1, 2
3: **repeat**
4:     Generate an episode using actions $a = M\tanh(\mu_\theta(s) + \epsilon)$ where $\epsilon \sim \mathcal{N}(0, \sigma_1)$.
5:     update $I_{recent}, I_{max}$ with training episode returns, let $K$ = length of episode
6:     compute $\eta = \eta_0 \cdot \frac{I_{recent}}{I_{max}} + (1 - \frac{I_{recent}}{I_{max}})$
7:     **for** $j$ in range($K$) **do**
8:         Compute $c_k = N \cdot \eta^{k\frac{1000}{K}}$
9:         Sample a batch of transitions, $B = \{(s, a, r, s)\}$ from most recent $c_k$ data in $\mathcal{D}$
10:        Compute targets for Q functions:
               $y_q(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', M\tanh(\mu_\theta(s') + \delta)) \qquad \delta \sim \mathcal{N}(0, \sigma_2)$
11:        Update Q-functions by one step of gradient descent using
               $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\phi,i}(s, a) - y_q(r, s'))^2$ for $i = 1, 2$
12:        Update policy by one step of gradient ascent using
               $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi,1}(s, M\tanh(\mu_\theta(s)))$
13:        Update target networks with
               $\phi_{\text{targ, i}} \leftarrow \rho\phi_{\text{targ, i}} + (1 - \rho)\phi_i$ for $i = 1, 2$
14:    **end for**
15: **until** Convergence
```

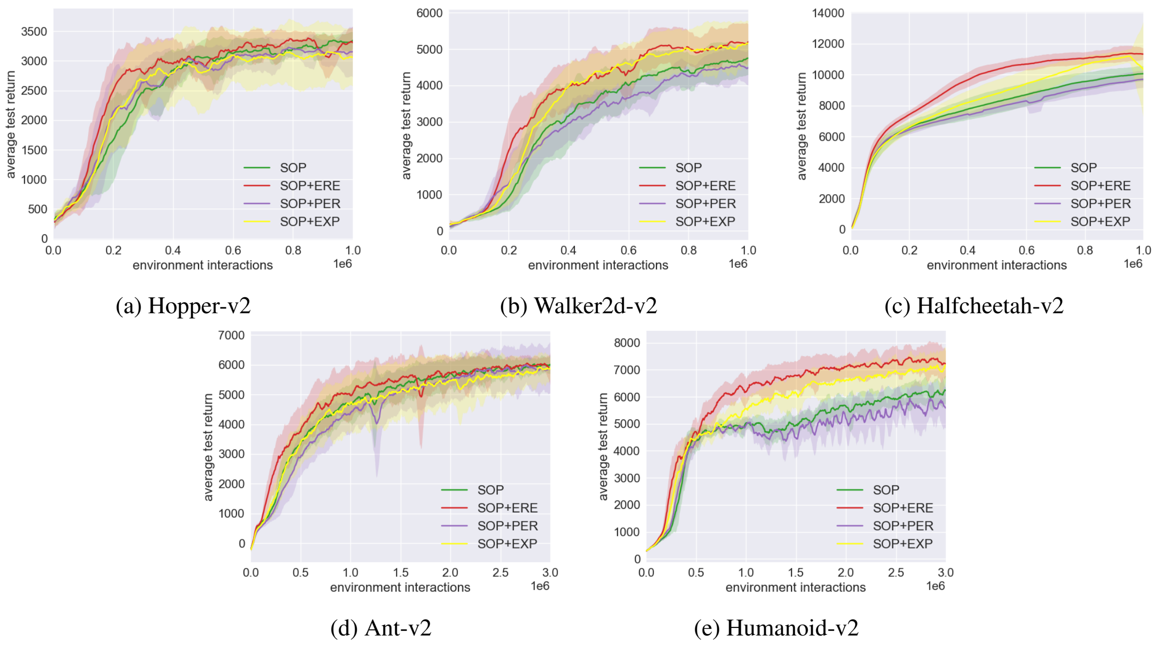


Figure 2: Streamlined Off-Policy (SOP), with ERE and PER sampling schemes

We apply this sampling scheme to SOP and refer to this variant as SOP+EXP.

5.3. PER and EXP experiment results

Figure 2 shows a performance comparison of SOP, SOP+ERE, SOP+EXP and SOP+PER. Results show that the exponential sampling scheme gives a boost to the performance of SOP, and especially in the Humanoid environment, although not as good as ERE. Surprisingly, SOP+PER does not give a significant performance boost to SOP (if any boost at all). We also found that it is difficult to find hyperparameter settings for SOP+PER that work well for all environments. Some of the other hyperparameter settings actually reduce performance. It is unclear why PER does not work so well for SOP. A similar result has been found in another recent paper (Fu et al., 2019), showing that PER can significantly reduce performance on TD3. Further research is needed to understand how PER can be successfully adapted to environments with continuous action spaces and dense reward structure.

6. Additional ERE analysis

Figure 3 shows, for fixed η, how η affects the data sampling process, under the ERE sampling scheme. Recent data points have a much higher probability of being sampled compared to older data, and a smaller η value gives more emphasis to recent data.

Different η values are desirable depending on how fast the agent is learning and how fast the past experiences become obsolete. So to make ERE work well in different environments with different reward scales and learning progress, we adapt η to the the speed of learning. To this end, define performance to be the training episode return. Define I_{recent} to be how much performance improved from $N/2$ timesteps ago, and I_{max} to be the maximum improvement throughout training, where N is the buffer size. Let the hyperparameter η_0 be the initial η value. We then adapt η according to the formula: $\eta = \eta_0 \cdot I_{recent}/I_{max} + 1 - (I_{recent}/I_{max})$.

Under such an adaptive scheme, when the agent learns quickly, the η value is low in order to learn quickly from new data. When progress is slow, η is higher to make use of the stabilizing effect of uniform sampling from the whole buffer.

7. Additional implementation details

7.1. ERE implementation

In this section we discuss some programming details. These details are not necessary for understanding the algorithm, but they might help with reproducibility.

In the ERE scheme, the sampling range always starts with the entire buffer (1M data) and then gradually shrinks. This is true even when the buffer is not full. So even if there are not many data points in the buffer, we compute c_k based as if there are 1M data points in the buffer. One can also modify the design slightly to obtain a variant that uses the current amount of data points to compute c_k. In addition to the reported scheme, we also tried shrinking the sampling range linearly, but it gives less performance gain.

In our implementation we set the number of updates after an episode to be the same as the number of timesteps in that episode. Since environments do not always end at 1000 timesteps, we can give a more general formula for c_k. Let K be the number of mini-batch updates, let N be the max size of the replay buffer, then:

$$c_k = N \cdot \eta^{k \frac{1000}{K}} \tag{6}$$

With this formulation, the range of sampling shrinks in more or less the same way with varying number of mini-batch updates. We always do uniform sampling in the first update, and we always have $\eta^{K \frac{1000}{K}} = \eta^{1000}$ in the last update.

When η is small, c_k can also become small for some of the mini-batches. To prevent getting a mini-batch with too many repeating data points, we set the minimum value for c_k to 5000. We did not find this value to be too important and did not find the need to tune it. It also does not have any effect for any $\eta \geq 0.995$ since the sampling range cannot be lower than 6000.

In the adaptive scheme with buffer of size 1M, the recent performance improvement is computed as the difference of the current episode return compared to the episode return 500,000 timesteps earlier. Before we reach 500,000 timesteps, we simply use η_0. The exact way of computing performance improvement does not have a significant effect on performance as long as it is reasonable.

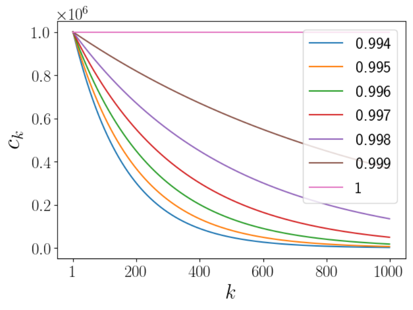
7.2. Programming and computation complexity

In this section we give analysis on the additional programming and computation complexity brought by ERE and PER.

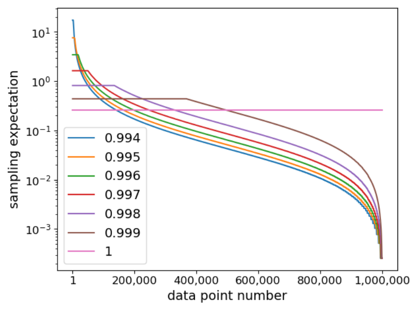
In terms of programming complexity, ERE is a clear winner since it only requires a small adjustment to how we sample mini-batches. It does not modify how the buffer stores the data, and does not require a special data structure to make it work efficiently. Thus the implementation difficulty is minimal. PER (proportional variant) requires a sum-tree data structure to make it run efficiently. The implementation is not too complicated, but compared to ERE it is a lot more work.

The exponential sampling scheme is very easy to implement, although a naive implementation will incur a significant computation overhead when sampling from a large buffer. To improve its computation efficiency, we instead uses an approximate sampling method. We first sample data indexes from segments of size 100 from the replay buffer, and then for each segment sampled, we sample one data point uniformly from that segment.

In terms of computation complexity (not sample efficiency), and wall-clock time, ERE's extra computation is negligible. In practice we observe no difference in computation time between SOP and SOP+ERE. PER needs to

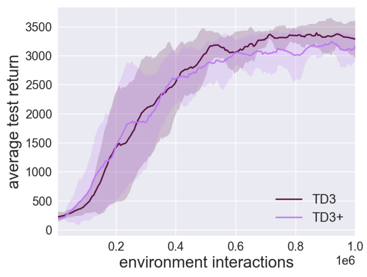


(a)

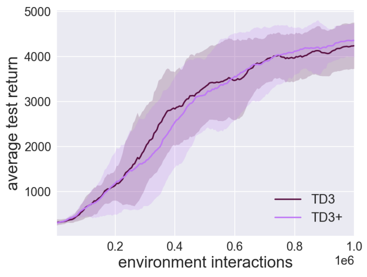


(b)

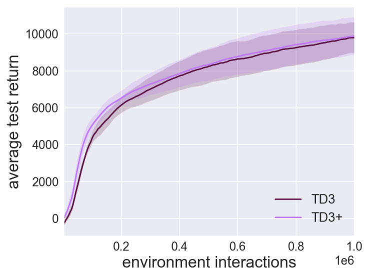
Figure 3: Effect of different η values. The plots assume a replay buffer with 1 million samples, and 1,000 mini-batches of size 256 in an update phase. Figure 3a plots c_k (ranging from 0 to 1 million) as a function of k (ranging from 1 to 1,000). Figure 3b plots the expected number of times a data point in the buffer is sampled, with the data points ordered from most to least recent.



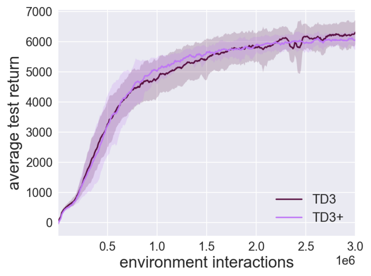
(a) Hopper-v2



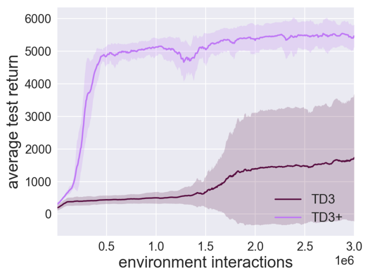
(b) Walker2d-v2



(c) HalfCheetah-v2



(d) Ant-v2



(e) Humanoid-v2

Figure 4: TD3 versus TD3+ (TD3 plus the normalization scheme)

update the priority of its data points constantly and compute sampling probabilities for all the data points. The complexity for sampling and updates is $O(log(N))$, and the rank-based variant is similar (Schaul et al., 2015). Although this is not too bad, it does impose a significant overhead on SOP: SOP+PER runs twice as long as SOP. Also note that this overhead grows linearly with the size of the mini-batch. The overhead for the MuJoCo environments is higher compared to Atari, possibly because the MuJoCo environments have a smaller state space dimension while a larger batch size is used, making PER take up a larger portion of computation cost. For the exponential sampling scheme, the extra computation is also close to negligible when using the approximate sampling method.

In terms of the proposed normalization scheme and the Inverting Gradients (IG) method, the normalization is very simple and can be easily implemented and has negligible computation overhead. IG has a simple idea, but its implementation is slightly more complicated than the normalization scheme. When implemented naively, IG can have a large computation overhead, but it can be largely avoided by making sure the gradient computation is still done in a batch-manner. We have made a very efficient implementation and our code is publicly available so that interested reader can easily reproduce it.

7.3. Computing Infrastructure

Experiments are run on cpu nodes only. Each job runs on a single Intel(R) Xeon(R) CPU E5-2620 v3 with 2.40GHz.

8. TD3 versus TD3+

In figure 4, we show additional results comparing TD3 with TD3 plus our normalization scheme, which we refer as TD3+. The results show that after applying our normalization scheme, TD3+ has a significant performance boost in Humanoid, while in other environments, both algorithms achieve similar performance.

References

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

Fu, J., Kumar, A., Soh, M., and Levine, S. Diagnosing bottlenecks in deep q-learning algorithms. *arXiv preprint arXiv:1902.10250*, 2019.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hausknecht, M. and Stone, P. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

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

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.