Ensemble Learning: Teaching Agents to Walk

Maria F. Harris
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, USA
m2harris@ucsd.edu

Roumen Guha
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, USA
roguha@ucsd.edu

***Abstract*—This paper presents a particular solution that builds upon twin-delayed deep deterministic policy gradients (TD3) to tackle the problem of motion planning for bipedal agents via reinforcement learning. Very briefly, we implemented something called Multi-TD3 where we train two separate instances of TD3 simultaneously (and share the same replay buffer), but at the end of every epoch we select the one with better evaluation returns to create the two instances for the next epoch, allowing us to sample more actions from the reply buffer. We achieve better rewards sooner, and faster training after the initial plateau. We compare with vanilla TD3 and SUNRISE to benchmark our performance. We evaluate these methods on the suite of OpenAI Gym environments.**

***Index Terms*—reinforcement learning, ensemble learning, actor-critic, Multi-TD3, TD3, DDPG, policy gradient, motion planning, walking, bipedal agent, robotics**

I. Introduction

It has been a long-time goal of the field of artificial intelligence to solve complex tasks from sensory data, without the need for much preprocessing between input and output. Reinforcement learning (RL) is a methodology that is able to result in such processes.

However, most model-free reinforcement learning algorithms suffer from requiring large amounts of training data–which translates to long training times or expensive parallel computational resources. These model-free algorithms do not attempt to build a model of the environment, and are therefore less sample-efficient compared to their model-based counterparts. Thus the engineering problem reduces to finding a learning methodology that generalizes well to a specific task.

Multiple algorithms have been released for this task, and we briefly discuss them later in the paper, to serve in contextualizing our work. However, the broad goal of this project was to propose a novel reinforcement learning network architecture to train a policy that can enable a bipedal agent to walk. To that end, we implement something we term Multi-TD3. Multi-TD3 is an ensemble learning method that trains two (or more) TD3 instances in every epoch. In Section II, we explain the mechanisms necessary to understand Multi-TD3 learning algorithm.

A. Related Work

In this section, we will briefly discuss other ensemble algorithms that have found some success in reinforcement learning.

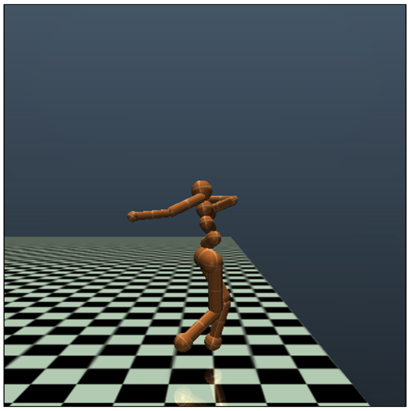


Fig. 1: Humanoid-V2 trained with Multi-TD3 for 1 million episodes. Snapshot was approximately taken at step 500 out of 1000, yet shows the agent almost outside of the environment. This speaks to how well the learned policy performs.

1) STEVE: In Buckman et al.'s work [1], the authors present Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion (STEVE). STEVE relies on the novel idea of attempting to find a uncertainty-aware balance between model-based and model-free learning. A model-based learning algorithm is one which learns a transition function f in order to estimate the optimal policy. This typically provides information about the environment that a model-free learning approach would only be able to approximate. Because of this difference, model-based approaches are able to reduce the number of samples required to learn a policy, thereby increasing the sample efficiency (often by several orders of magnitude).

STEVE, by using rollouts to improve the targets for the temporal difference (TD) learning, is able to use information gleaned from these rollouts as inputs to the policy. However, in dynamic, complex, or noisy environments it can be difficult to learn a policy that performs well in the environment using model-based learning. This is the primary obstacle when using model-based learning. Buckman et al. address this in STEVE by replacing the model and the Q-function with ensembles. This interpolates between different horizon lengths, and favors those whose estimates are small (reducing the uncertainly and error introduced from them).

From their experiments, it seems as though STEVE was able to at least match the performance of DDPG, and more often beat it.

2) ME-TRPO: Kurutach et al. [2] present something they call Model-Ensemble Trust-Region Policy Optimization (ME-TRPO). ME-TRPO is a purely model-based policy-gradient approach, and uses the ensemble to avoid overfitting to any one model. Briefly, ME-TRPO integrates the following three components:

- **Model ensemble.** Kurutach et al. fit a set of dynamics models $\{f_{\phi_1}, \dots, f_{\phi_K}\}$ using the same data we sample for learning the policy. These are each trained to predict the change in state given a state and an action as inputs. The output is then added to the input state to approximate the next state. The functions f_{ϕ_i} differ only in their randomly initialized weights and the order in which the mini-batches are sampled.
- **TRPO.** Use TRPO (as described in [3]) to optimize the policy over the model ensemble. In every step, randomly choose a model to predict the next state given the current state and action.
- **Early stopping.** Use the model ensemble to monitor the performance of the policy on the validation set. Use early stopping on the current iteration if the policy stops improving.

The understanding behind these steps is that the model ensemble should serve as effective regularization for policy learning, thus avoiding overfitting by forcing the policy to perform well over several distinct possible future trajectories. As they found in their results, the more models used in the model ensemble, the better the regularization and the better the performance. They were able to perform better than TRPO and DDPG in every environment they attempted.

II. Background

This section describes the notation, the problem being addressed by our proposed method, and the background information necessary to understand our method.

In reinforcement learning, we have an agent that interacts with its environment, and its main goal is to learn certain behaviours that will maximize its rewards. At each discrete timestep t, the agent receives observations from the environment known as states $s_t \in \mathcal{S}$, and performs an action $a \in \mathcal{A}$ according to its policy $\pi(s)$ for which it receives its reward r and the new state of its environment s_{t+1}. The $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is defined as the cumulative rewards from timestep t, where $\gamma \in [0, 1)$ is known as the discount factor.

The goal of reinforcement learning is to find the optimal policy π_θ which maximizes the expected return: $J(\theta) = \mathbb{E}_{\pi_\theta}[R_t]$. For continuous spaces, the policy π_θ can be updated by taking the gradient of the expected return: $\nabla_\theta J(\theta)$. An actor-critic method utilizes both policy optimization and Q-learning; therefore, for actor-critic methods with a deterministic policy (also known as the actor), π_θ, the policy can be updated using the deterministic policy gradient algorithm [4] shown in Eq. 1. The critic is responsible for evaluating the current policy.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p_\pi}[\nabla_a Q_\pi(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)] \tag{1}$$

Where, $Q_\pi = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi_\theta}[R_t | s, a]$ is the expected return when doing an action a with state s using its policy π_θ. This is known as the critic, and is responsible for evaluating the value function.

For estimating the value function, also known as Q-learning, the estimation can be done using an update rule based on the Bellman equation [5]:

$$Q_\pi(s,a) = r(s,a) + \gamma\, \mathbb{E}_{s',a'}[Q_\pi(s',a')] \tag{2}$$

For deep Q-learning the model, is updated using temporal difference learning [6] in conjunction with a secondary target network $Q'_\phi(s,a)$ to maintain a fixed objective y over numerous updates:

$$y = r(s,a) + \gamma Q_{\phi'}(s',a') \tag{3}$$

Where, the target action a' is selected from a target actor model, and $Q_{\phi'}$ is a differentiable function parameterized over ϕ.

A. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient is a model free algorithm which learns both the policy and a Q-function (actor-critic model) [7]. It relies on off-policy data and the bellman equation to estimate the Q-function, and then uses this Q-function for learning the policy.

DDPG employs two techniques: replay buffers, and target networks.

Experience replay buffers store the previous experiences and samples are drawn from it for updating the critic and actor networks. Experience buffer replays prevent overfitting since batches of samples are drawn from it at random.

Target networks as shown in Eq. 3. It is called the target network since when we minimize the loss, we are trying to make the Q-function equivalent to this target network.

1) Q-Function Estimation: The Q-function is estimated using Eq. 2. Using the estimated Q-function, we can then find and minimize the mean-squared Bellman error function using the target y defined in Eq. 3 as follows:

$$L(\phi) = \mathbb{E}_{s,a,s',a'}[(Q_\phi(s,a) - y)^2] \tag{4}$$

2) Policy Learning: DDPG learns a deterministic policy π_θ. Since the action space is continuous, and Q_ϕ is differentiable, we can therefore perform gradient ascent mentioned in eq 1 to solve the following:

$$\max_\theta \mathbb{E}_s[Q_\phi(s, \pi_\theta(s))] \tag{5}$$

B. Twin Delayed Deep Deterministic Policy Gradient

The Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is an off-policy algorithm, and is the successor to the Deep Deterministic Policy Gradient (DDPG) algorithm. It mainly addresses the issue faced by DDPG: the Q-function starts overestimating the Q-values, which leads to the policy breaking since it exploits the errors in the Q-function [4]. TD3 mitigates these issues by applying the following modifications: 1) Target Policy Smoothing, 2) Delayed Policy Updates and 3) Clipped Double Q-Learning. By building upon TD3, we inherit these properties of TD3 into Multi-TD3.

1) Target Policy Smoothing: TD3 adds noise to the target action a' to make it more difficult for the policy take advantage of the Q-function errors by smoothing out the Q-function. Mathematically, this is represented as follows:

$$y = r(s,a) + \gamma Q_{\phi'}(s', \pi_{\theta'}(s') + \epsilon) \tag{6}$$

Where, $\epsilon \sim clip(\mathcal{N}(0,\sigma), -k, k)$

2) Clipped Double Q-Learning: Instead of learning one Q-function, TD3 learns two Q-functions. The minimum of the two Q-values is selected to give the single target as follows:

$$y = r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \pi_\theta(s')) \tag{7}$$

Using a target function as the smallest of the two Q-values ensures that there will be no overestimation in the Q-function.

3) Delayed Policy Updates: Delaying the policy updates decreases the probability of repeating values when the critic is unchanged. Due to the delayed policy updates, the repeated values that do occur will use a value estimate with lower variance which in turn would produce better policy updates.

C. SUNRISE

We compare our Multi-TD3 with **S**imple **UN**ified framework for **R**e**I**nforcement learning using en**SE**mbles (SUNRISE). Like Multi-TD3, SUNRISE [8] is an off-policy ensemble reinforcement learning algorithm. It is different in its formulation; roughly, it integrates three main components:

- **Bootstrap with random initialization.** Bootstrapping is used here to enforce diversity in the training samples the agents experience. The random initialization helps to prevent bias towards the initial values.
- **Weighted Bellman backups.** The authors use the variance of predictions from the agents to characterize uncertainty estimates for their target Q-function. They use these uncertainty estimates to weight updates to the Bellman equation, which leads to a significant reduction in error propagating into their current Q-function.
- **Upper-confidence bound action selection.** In [8], Lee et al. formalize an expression for what they call the upper-confidence bound (UCB) that incorporates information based on the mean and variance of the Q-functions. They use this to create an inference method for action selection, which introduces a new possibility for exploration

Algorithm 1: Multi-TD3

```
Input: initialize policy paramaters $\theta$, Q-function
  parameters $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$
Assign parameters to two instances of the TD3 model:
  $M_1$ and $M_2$
Set target paramaters equal to main parameters
  $\theta_{targ} \leftarrow \theta$, $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$
repeat
    Do for both models:
    Observe state $s$ and select action $a$ =
      clip($\mu_\theta(s) + \epsilon$, $\alpha_{Low}$, $\alpha_{High}$), where $\epsilon \sim \mathcal{N}$
    Execute $a$ in the environment
    Observe next state $s'$, reward $r$, and done signal $d$
      to indicate whether $s'$ is terminal
    Store ($s$, $a$, $r$, $s'$, $d$) in replay buffer $\mathcal{D}$
    if $s'$ is terminal then reset environment state;
    if it's time to update then
        for j in range (however many updates) do
            Randomly sample a batch of transitions, B
              = {($s$, $a$, $r$, $s'$, $d$)} from $\mathcal{D}$
            Compute target actions:
              $a'(s') = \text{clip}(\mu_\theta(s)+\epsilon, \alpha_{Low}, \alpha_{High}), \quad \epsilon \sim \mathcal{N}$
            Compute targets:
              $y(r,s',d) = r+\gamma(1-d)\min_{i=1,2} Q_{\phi_{targ,i}}(s', a'(s'))$
            Update Q-functions by one step of gradient
              descent using
              $\nabla_{\phi_i}\frac{1}{|B|}\Sigma_{(s,a,r,s',d)\in B}(Q_{\phi_i}(s,a)-y(r,s',d))^2$
              for $i = 1, 2$
            if j mod policy_delay = 0 then
                Update policy by one step of gradient
                  ascent using
                  $\nabla_{\phi_i}\frac{1}{|B|}\Sigma_{s\in B}(Q_{\phi_1}(s,\mu_\theta(s)))$
                Update target networks with
                  $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1-\rho)\phi_i$
                  $\theta_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\theta$
                  for $i = 1, 2$
    Select the model which has the largest returns and
      assign it to both $M_1$ and $M_2$.
until convergence;
```

by providing a bonus for previously-unseen state-action pairs.

SUNRISE was implemented over SAC for the continuous version, which is described in [9].

III. Methodology

This section provides an overview of the utilized framework and introduces the proposed model, Multi-TD3.

A. Agent's Framework

For training and testing the proposed model, we used OpenAI's Gym [10]. Gym is an open source interface which is used for performing RL-based tasks. We used three environments from Gym: Ant-v2, HalfCheetah-v2, and Walker2D-v2. Below are the observation and action space for both the environments:

- **Walker2D-v2:**
  - Observation Space: $s \in \mathbb{R}^{17}$
  - Action Space: $a \in \mathbb{R}^{6}$
- **HalfCheetah-v2:**
  - Observation Space: $s \in \mathbb{R}^{17}$
  - Action Space: $a \in \mathbb{R}^{6}$
- **Ant-v2:**
  - Observation Space: $s \in \mathbb{R}^{111}$
  - Action Space: $a \in \mathbb{R}^{8}$

We trained on the environments as they are; we did not make any modifications to the rewards functions.

B. Multi-TD3

Multi-TD3 is an off-policy algorithm based on the original Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm [4]. Multi-TD3 works on the principle of ensemble learning. As a whole, Multi-TD3 performs the following addition steps in comparison to the vanilla TD3 algorithm:

- Two separate instances of TD3 networks are initialized along with two instances of the environment with the same agent.
- During training, both the instances of TD3 push data into the experience replay buffer, and draw samples from it, randomly.
- For each complete episode, the instance which has the largest total rewards is selected, and this model then serves as the model to be trained for the next complete episode by each of the two instances of TD3.

The main advantage of the multi-TD3 is that this technique reduces the likelihood of selecting a poor model since the model with the highest rewards is always chosen after a complete episode, this ensures that only the best model can be trained during the next episode. Moreover, the experience replay buffer is also utilized more effectively since both the models push and sample experiences from it. Therefore, we sampling more actions per episode, leading to greater rewards.

We have provided pseudo code in Alg. 1. We trained using Adam as the optimizer [11].

IV. Results

We found that we could not compare SUNRISE and vanilla-TD3 on the same figure; SUNRISE takes a prohibitively long time to train compared to vanilla-TD3 and Multi-TD3 for the same number of episodes. Because of this, we have one set of figures over fewer timesteps to compare Multi-TD3 and SUNRISE, and another set of figures to compare Multi-TD3 and vanilla-TD3. The training curves can be seen in Fig. 2.

From the training curves, we are able to make a few interesting observations. First, SUNRISE easily outperforms Multi-TD3. Second Multi-TD3 makes noticeable but not exactly significant improvements over vanilla-TD3.

Otherwise, we are basically guaranteed a better result through Multi-TD3 compared to vanilla-TD3.

The results can be seen in Table I and Table II.

V. Discussion

In this project, we were able to create and test an ensemble reinforcement learning method against three OpenAI Gym physical simulation environments. We compared the performance of our algorithm across these environment against the algorithm we built upon, and another state-of-the-art algorithm to show the difference between them.

A. Limitations

Because SUNRISE proved as difficult to train as it did, we could not attempt to train it on the Humanoid-V2 environment as we initially intended on doing, and therefore the environment couldn't be used for comparison in the results section. We did find that Multi-TD3 performed noticeably and consistently better than vanilla-TD3 did, and was easily able to complete the Humanoid-V2 environment. Please see a screenshot of the walking humanoid in Fig. 1.

B. Impact

Recent progress in deep reinforcement learning has been remarkable, but the core issues in training an RL algorithm still remain:

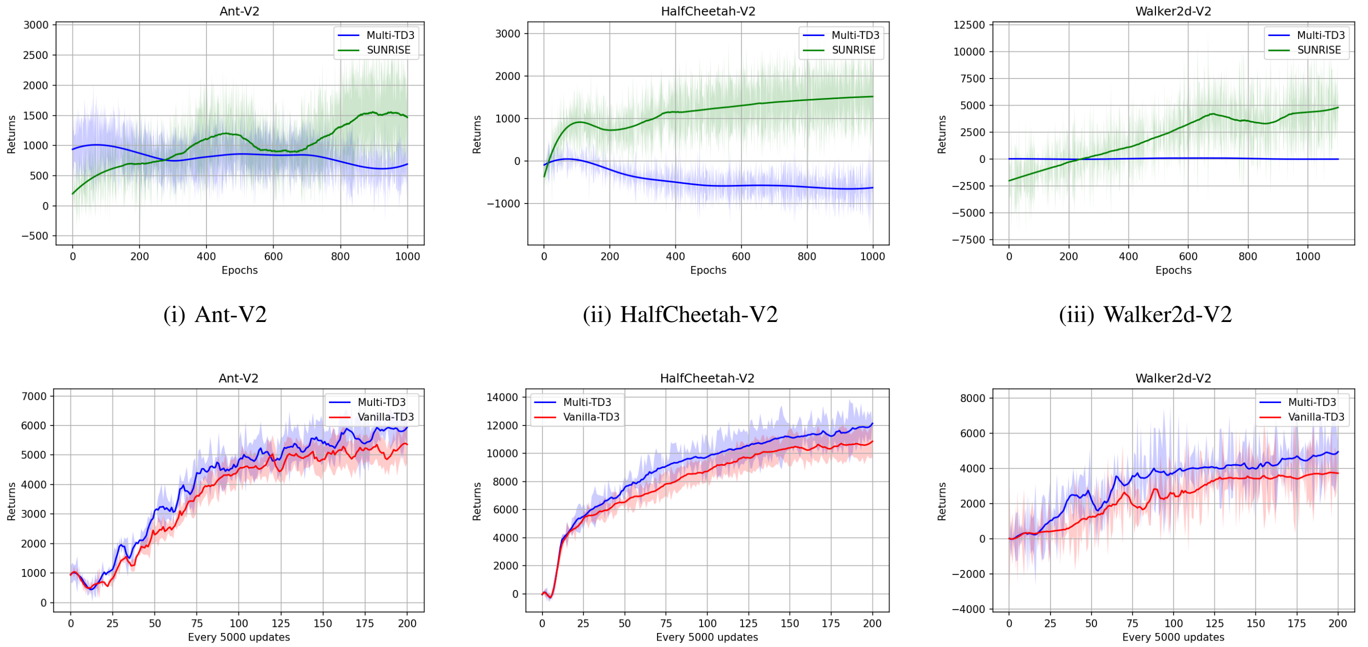
- Model-free learning is still highly sample-inefficient.

Method	**Ant-v2**	**HalfCheetah-v2**	**Walker2d-v2**
Multi-TD3	1011.1 ± 107.0	50.4 ± 22.8	385.6 ± 40.0
SUNRISE	1556.0 ± 333.2	1520.1 ± 342.7	4795.4 ± 204.4

TABLE I: Experimentation results. Max average return over a thousand episodes, with ± corresponding to a single standard deviation over trials. Each experiment was run three times.

Method	**Ant-v2**	**HalfCheetah-v2**	**Walker2d-v2**
Multi-TD3	5940.8 ± 171.5	12139.5 ± 292.7	4940.6 ± 142.2
Vanilla-TD3	5392.6 ± 163.0	10862.6 ± 267.3	3766.5 ± 122.9

TABLE II: Experimentation results. Max average return over a million episodes, with ± corresponding to a single standard deviation over trials. Each experiment was run three times.



(i) Ant-V2

(ii) HalfCheetah-V2

(iii) Walker2d-V2

(iv) Ant-V2

(v) HalfCheetah-V2

(vi) Walker2d-V2

Fig. 2: Returns vs timestep. The row above compares Multi-TD3 (blue) to SUNRISE (green) over a thousand timesteps. The row below compares Multi-TD3 (blue) to Vanilla-TD3 (red) over a million timesteps. Curves were smoothed using the Savitzky–Golay filter.

- The training process is still seemingly brittle to changes in hyperparameters, and it is fairly opaque as to what modifications will lead to improvements due to the non-linear nature of the problems.
- Learning algorithms can suffer from problems with random initialization, and become unable to overcome local minimums. Due to this, we can converge to sub-optimal solutions. However, this is a trade-off: too much exploration will lead to too little exploitation, meaning our agent will not learn an optimal policy either.

Ensemble methods, like Multi-TD3 and SUNRISE, are a class of learning methods which seek to stabilize the learning process while still providing enough exploration for the agent to exploit. Due to the nature of ensemble methods, it should be fairly easy to extend the principles found in Multi-TD3 and SUNRISE to other learning algorithms. Thus, we believe that the simple idea behind Multi-TD3 could be adapted to other algorithms, and become relevant to topics such as motion planning in robotics, and particularly in topics such as hot-air balloon navigation while conserving power, or drone navigation.

C. Future Work

In this work, we compared Multi-TD3 with TD3 and SUNRISE. Due to this, we kept some hyperparameters consistent between algorithms, such as the size of the replay buffer and the number of past trajectories sampled from the replay buffer. To this end, we cannot say if we were comparing each learning algorithm's best performance to one another;it may be possible that these hyperparameters change depending on how the information is used. This should be the work of a future experiment.

There are other minor modifications that would likely lead to improvements as well. For example, it is possible to prioritize the samples in the replay buffer by their return values; leading to higher sampling of actions that have typically led to better rewards [12].

Another potential, relatively minor modification that could be implemented is the addition of normalized parameter-space noise [13].

Acknowledgments

Thank you for taking the time to read this paper. We would like to take this time to acknowledge how much work must go into this course, and how difficult it can be to manage resources for several projects simultaneously. We would like to tell you that we are grateful for these opportunities and your feedback throughout the project.

References

[1] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "STEVE: Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion," jul 2018. [Online]. Available: http://arxiv.org/abs/1807.01675

[2] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv*, pp. 1–15, 2018. [Online]. Available: https://arxiv.org/pdf/1802.10592

[3] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," feb 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[4] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," feb 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

[5] R. E. Bellman, *Dynamic Programming*, 1st ed. Princeton: Princeton University Press, 1957. [Online]. Available: http://books.google.com/books?id=fyVtp3EMxasC{&}pg=PR5{&}dq=dynamic+programming+richard+e+bellman{&}client=firefox-a{#}v=onepage{&}q=dynamicprogrammingrichardebellman{&}f=false

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: https://www.google.com/books/edition/Reinforcement{_}Learning{_}second{_}edition/uWV0DwAAQBAJ?hl=en{&}gbpv=1{&}dq=info:t8N5xiW9bXoJ:scholar.google.com{&}pg=PR7{&}printsec=frontcover

[7] 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," sep 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[8] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, "SUNRISE: A Simple Unified Framework for Ensemble Learning in Deep Reinforcement Learning," jul 2020. [Online]. Available: http://arxiv.org/abs/2007.04938

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," jan 2018. [Online]. Available: http://arxiv.org/abs/1801.01290

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," dec 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[12] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," nov 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[13] 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," jun 2017. [Online]. Available: http://arxiv.org/abs/1706.01905