Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent

Conference Paper · August 2019

DOI: 10.1145/3387168.3387199

2 authors:

Stephen Dankwa
University of Electronic Science and Technology of China
8 PUBLICATIONS **262** CITATIONS
SEE PROFILE

Wenfeng Zheng
University of Electronic Science and Technology of China
216 PUBLICATIONS **9,985** CITATIONS
SEE PROFILE

Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent

Stephen Dankwa
Automation Engineering
University of Electronic Science and Technology of China
nabistephen@yahoo.com

Wenfeng Zheng
Automation Engineering
University of Electronic Science and Technology of China
winfirms@uestc.edu.cn

ABSTRACT

In this current research, Twin-Delayed DDPG (TD3) algorithm has been used to solve the most challenging virtual Artificial Intelligence application by training a 4-ant-legged robot as an Intelligent Agent to run across a field. Twin-Delayed DDPG (TD3) is an incredibly smart AI model of a Deep Reinforcement Learning which combines the state-of-the-art methods in Artificial Intelligence. These includes Policy gradient, Actor-Critics, and continuous Double Deep Q-Learning. These Deep Reinforcement Learning approaches trained an Intelligent agent to interact with an environment with automatic feature engineering, that is, necessitating minimal domain knowledge. For the implementation of the TD3, we used a two-layer feedforward neural network of 400 and 300 hidden nodes respectively, with Rectified Linear Units (ReLU) as an activation function between each layer for both the Actor and Critics. We, then added a final tanh unit after the output of the Actor. The Critic receives both the state and action as input to the first layer. Both the network parameters were updated using Adam optimizer. The idea behind the Twin-Delayed DDPG (TD3) is to reduce overestimation bias in Deep Q-Learning with discrete actions which are ineffective in an Actor-Critic domain setting. Based on the Maximum Average Reward over the evaluation time-step, our model achieved an approximate maximum of 2364. Therefore, we can truly say that, TD3 has obviously improved on both the learning speed and performance of the Deep Deterministic Policy Gradient (DDPG) in a challenging environment in a continuous control domain.

CCS Concepts

•Computing methodologies → Modeling and simulation → Model development and analysis → Model verification and validation

Keywords

Deep Reinforcement Learning; Artificial Intelligence; Twin-Delayed Deep Deterministic Policy Gradient; Actor-Critic

ICVISP 2019, August 26–28, 2019, Vancouver, BC, Canada
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7625-9/19/08…$15.00
https://doi.org/10.1145/3387168.3387199

1. INTRODUCTION

Recently, Deep Reinforcement Learning has gain tremendous attention in the machine learning community as a result to the essential amount of progress that has been achieved for the past few years. Reinforcement Learning is the ability of an agent to learn what to do, how to map situations to actions, in order to maximize a numerical reward signal [12]. Figuring out how to deal with continuous action spaces was one of the subsequent challenges that the Reinforcement Learning community faced. Solving complex tasks from unprocessed, high-dimensional, input sensory is one of the primary aims in the field of Artificial Intelligence [5]. Deep Q-Network algorithms have been attempted and tried to solve problems with high-dimensional observation spaces but, however, it is able to handle discrete and low-dimensional action spaces [5]. Addition, several practical and physical control tasks are characterized with continuous and high-dimensional spaces. And as a result, Deep Q-Network cannot be directly applied to continuous domain settings since it depends on a result the action which maximizes the action-value function. The reason is that continuous valued-based request an iterative optimization process at every step [5]. In the course of solving the continuous and high-dimensional spaces problem, evolved the Deterministic Policy Gradient (DPG) algorithms [11], which is an off-policy actor-critic algorithm that learns a deterministic target policy. In their work, they demonstrated that deterministic policy Gradient algorithms significantly outperformed their stochastic counterparts in high-dimensional spaces. In addition, Deep Deterministic Policy Gradient (DDPG) algorithms [5] also emerged in the process of solving the continuous and high-dimensional spaces problem. The DDPG algorithm was built upon the DPG algorithm, which is also an off-policy actor-critic algorithm that uses deep function approximators that can learn policies in high-dimensional, continuous action spaces [5]. In their work, they stated that they combined the actor-critic technique with intuitions from the recent success of the Deep Q-Network (DQN). Deep Q-Network (DQN) [6] has the capability to learn value functions based on function approximators in a stable and robust manner as a result to two novelties [5]:

- The network is trained off-policy with samples from a replay buffer to minimize correlations between samples
- The network is trained with a target Q-Network to give reliable targets during temporal difference backups.

Furthermore, based on the success works from the Deep Q-Network (DQN) [6], the Deterministic Policy Gradient (DPG) [11] and Deep Deterministic Policy Gradient (DDPG) [5] algorithms paved way to the Twin-Delayed DDPG (TD3) algorithms [10]. Twin-Delayed Deep Deterministic Policy Gradient (TD3) is an off-policy model and recently has achieved breakthrough in Artificial Intelligence state-of-the-art models with continuous high-dimensional spaces. Reinforcement Learning algorithms that are characterized as off-policy generally utilized a separate behavior policy which is independent of the policy which is being improved upon. The key advantage of the separation is that the behavior policy can operate by sampling all actions, while the estimation policy can be deterministic [12].

Twin-Delayed Deep Deterministic Policy Gradient algorithm (TD3) was built on the Deep Deterministic Policy Gradient algorithm (DDPG) to increase stability and performance with consideration of function approximation error [10]. The uniqueness about the TD3 algorithm is in the combination of three powerful Deep Reinforcement Learning techniques such as continuous Double Deep Q-Learning [13], Policy Gradient [11] and Actor-Critic [7]. Most at times, the Researcher had tried to experiment the algorithm with a MuJoCo [5,6,10,11] continuous control tasks but, we experimented with the pybullet continuous control environment.

Therefore, in this current research, we applied the TD3 algorithm to model and train a 3D 4-Legged Ant robot in a continuous high-dimensional action space. We experimented with the pybullet continuous control environment which was interfaced through the OpenAI Gym.

The three main ideas behind this current research are to (1) present the evolution process of the TD3 model, (2) develop a model using the TD3 algorithm, and (3) evaluate its performance. The importance of this current research is to present to the Artificial Intelligence community how Twin-Delayed Deep Deterministic Policy Gradient algorithm (TD3) algorithm has been applied to solve the most challenging virtual Artificial Intelligence application by training a 3D 4-Legged-Ant robot as Intelligent Agent to walk and run across a field.

2. METHODOLOGY

Twin-Delayed DDPG (TD3) consists of three Deep Reinforcement Learning methods such the continuous Double Deep Q-Learning, Policy Gradient and Actor-Critic. This section presents the experimental details, the TD3 algorithm and the steps used in implementing the model.

2.1 Experiment Details

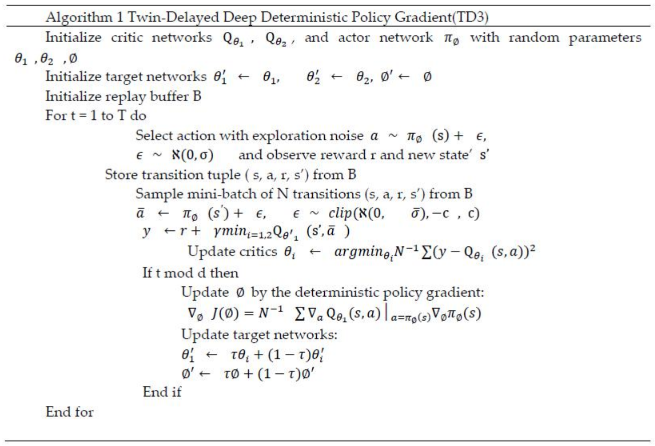
We implemented the TD3 model by using two-layer feedforward neural network of 400 and 300 hidden nodes respectively. We used Rectified Linear Units (ReLU) as an activation function between each layer for both the Actor and Critics, and then a final tanh unit following the output of the Actor. The Critic receives both the state and action as input to the first layer. The parameters set and utilized in the model were the name of the environment (AntBulletEnv-v0), random seed number (0), start timesteps (10000), evaluation steps (5000), maximum timesteps (500000), save model (true), exploration noise (0.1), batch size (100), discount factor (0.99), target network update rate (0.005), policy noise (0.2), clip noise (0.5) and policy frequency (2). Both the network parameters were updated using Adam optimizer [4]. The environment was achieved using the pybullet [2] continuous control tasks. Pybullet is a python module for physics stimulation, robotics, and reinforcement learning based on the Physics Bullet SDK. We interfaced the pybullet continuous control task, which is the environment (AntBulletEnv-v0) through the OpenAI Gym [1]. The parameters and their values used in this research work are been summarized in Table 1. The TD3 algorithm can be seen from Algorithm 1.

In this work, we used the Python version 3 environment as the programming language. We wrote and executed our code using the Jupyter notebook through the Google Colaboratory (https://colab.research.google.com/). Google Colaboratory is a free jupyter notebook that requires no setup and execute completely in the cloud. It offered us a free virtual machine server with GPU to execute the code.

The only package that was installed was the pybullet version 2.5.3. The rest of the libraries that were utilized in this work were, the numpy, gym, and torch. These libraries were imported directly from the google colab platform without installations.

Table 1. The Parameters of the TD3 model.

Parameter	Value	Description
Env_name	AntBulletEnv-v0	The name of the PyBullet environment
seed	0	Random seed number
Eval_freq	5e3	The number of times the evaluation step is performed
Max_timesteps	5e5	Total number of iterations
Save_model	true	Boolean checker, whether or not to save the pre-trained model
Expl_noise	0.1	Exploration noise
Bach_size	100	Size of the batch
discount	0.99	Discount factor gamma
Noise_clip	0.5	Maximum value of the Gaussian noise
Policy_freq	2	Number of iterations to wait before the policy network


```
Algorithm 1 Twin-Delayed Deep Deterministic Policy Gradient(TD3)
Initialize critic networks $Q_{\theta_1}$ , $Q_{\theta_2}$ , and actor network $\pi_\emptyset$ with random parameters $\theta_1$ ,$\theta_2$ ,$\emptyset$
Initialize target networks $\theta'_1 \leftarrow \theta_1$,   $\theta'_2 \leftarrow \theta_2$, $\emptyset' \leftarrow \emptyset$
Initialize replay buffer B
For t = 1 to T do
        Select action with exploration noise $a \sim \pi_\emptyset(s) + \epsilon$,
        $\epsilon \sim \aleph(0,\sigma)$ and observe reward r and new state' s'
    Store transition tuple ( s, a, r, s') from B
        Sample mini-batch of N transitions (s, a, r, s') from B
        $\bar{a} \leftarrow \pi_\emptyset(s') + \epsilon$,   $\epsilon \sim clip(\aleph(0, \bar{\sigma}), -c , c)$
        $y \leftarrow r + \gamma min_{i=1,2} Q_{\theta'_1}(s', \bar{a})$
            Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$
        If t mod d then
            Update $\emptyset$ by the deterministic policy gradient:
            $\nabla_\emptyset J(\emptyset) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\emptyset(s)} \nabla_\emptyset \pi_\emptyset(s)$
            Update target networks:
            $\theta'_1 \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
            $\emptyset' \leftarrow \tau\emptyset + (1-\tau)\emptyset'$
        End if
End for
```

2.2 Steps Used in Implementing the TD3 Algorithm

In this current research work, the steps that were implemented based on the TD3 algorithm can be followed as:

STEP 1: Initialize the Experience Replay Memory with a size of

20000 which will then be populated with each new transition

STEP 2: Build one neural network for the Actor model and one neural network for the Actor target

STEP 3: Build two neural networks for the two critic models and two neural networks for the two critic targets

STEP 4: Sample a batch of transitions (s, s', a, r) from the memory, where s=state, s'=next state, a=action, r=reward.

Then for each element of the batch:

STEP 5: From the next state s' , the Actor target plays the next action a'.

STEP 6: Add Gaussian noise to this next action a' and clamp it in a range of values supported by the environment.

STEP 7: The two Critic targets then take each of the couple (s', a') as input and return two Q-values

$Q_{t1}(s', a')$ and $Q_{t2}(s', a')$ as outputs.

STEP 8: Keep the minimum of these two Q-values as $\min(Q_{t1}, Q_{t2})$, which represents the approximated value of the next state.

STEP 9: Get the final target of the two Critic models, which is $Q_t = r + \Upsilon * \min(Q_{t1}, Q_{t2})$, where Υ is a discount factor.

STEP 10: The two Critic models take each of the couple (s, a') as input and return two Q-values

$Q_1(s, a')$ and $Q_{t2}(s, a')$ as outputs.

STEP 11: Then compute the Loss coming from the two Critic models.

Critic Loss = MSE_Loss($Q_1(s, a')$, Q_t) + MSE_Loss($Q_2(s, a')$, Q_t)

STEP 12: Backprobagate the Critic Loss and then update the parameters of the two Critic models with Stochastic Gradient Descent (SGD) optimizer

STEP 13: Once every two iterations, update the Actor model by performing Gradient Ascent on the output of the first Critic model: $\nabla_{\emptyset} J(\emptyset) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a) \big|_{a=\pi_{\emptyset}(s)} \nabla_{\emptyset}\pi_{\emptyset}(s)$, where $\emptyset$ and θ_1 are the weights of the Actor and the Critic respectively.

STEP 14: Once again at every two iterations, update the weights of the Actor target by Polyak Averaging:

$$\theta_1^{'} \leftarrow \tau\theta_i + (1-\tau)\theta_i^{'}$$

STEP 15: Once again at every two iterations, update the weights of the Critic target by Polyak Averaging:

$$\emptyset^{'} \leftarrow \tau\emptyset + (1-\tau)\emptyset^{'}$$

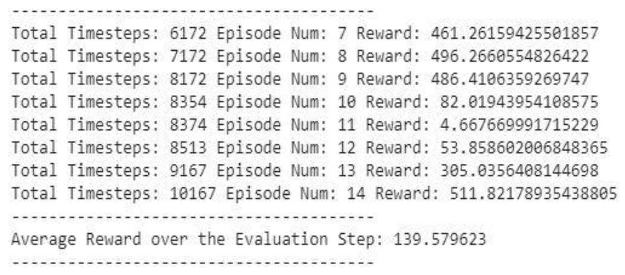
3. RESULTS AND DISCUSSIONS

In this current research work, we have modeled and trained an Intelligent 3D 4-Legged-Ant robot to interact with a continuous control environment to walk and run across a field using a current state-of-the-art Twin-Delayed DDPG algorithm. The main objectives were to present the evolution of the TD3 algorithm, develop a model using the TD3 algorithm and evaluate its performance with other state-of-the-art models. Within 10,000 iterations, we allowed the agent to explore on its own without training. After the exploration, the agent achieved an Average Reward of approximately 140, which can be seen from (figure 1). This Average Reward value was detected at the end of the 10,000 iterations.

After exposing the Agent to the training, the Agent then learned from the past pattern of the model. The agent improved upon its performance by achieving an approximate Average Reward of 2272 as seen from (figure 2) after the end of the 500,000 iterations. We observed a Maximum Average Reward of approximately 2364 at around 450,323 iterations as seen from (figure 3). In the original paper [10], in order to evaluate their TD3 algorithm, they used the MuJoCo continuous control task (Ant-v1) and interfaced it through the OpenAI Gym. In their experiment, they exposed the agent to training by performing 1,000,000 iterations. Their TD3 model then achieved a Maximum Average Reward of approximately 4372. The result of their model outperformed other start-of-the-art models such as DDPG [5], PPO [9], TRPO [8], ACKTR [14] and SAC [3] as seen from their work in Table 1[10].

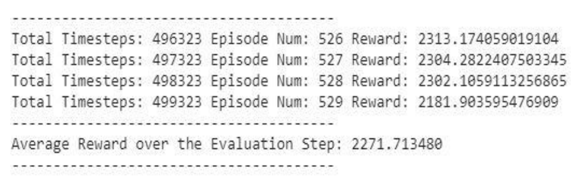
Whereas in this current research, we experimented with the pybullet continuous control task (AntBulletEnv-v0) (figure 4) and interfaced it through the OpenAI Gym. Apart from the continuous control task (AntBulletEnv-v0) and the number of iterations (500,000) which were changed, we maintained all the other parameters as in [10]. We exposed the agent to training by performing 500,000 iterations because of limited amount time we had when using a virtual machine server through the google colaboratory. Our current model then achieved a Maximum Average Reward of approximately 2364, Which makes sense, since we performed half of the iteration of the original paper. Our current model was as effective as the original paper when compared to other state-of-the-art models such as Deep Deterministic Policy Gradient (DDPG) [5], Proximal policy optimization algorithms (PPO) [9], Trust region policy optimization (TRPO) [8], Actor-Critic using using Kronecker-factored Trust Region (ACKTR) [14], and Soft Actor-Critic (SAC) [3].

The output of the model was a video (upon request can be released), representing the virtual AI application where the 3D 4-Legged-Ant walk and run across a field in a continuous control environment as in (figure 5). The direction of the movement was from left to right.



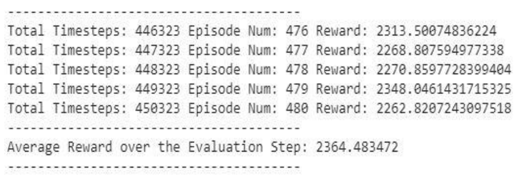
```
---------------------------------------
Total Timesteps: 6172 Episode Num: 7 Reward: 461.26159425501857
Total Timesteps: 7172 Episode Num: 8 Reward: 496.2660554826422
Total Timesteps: 8172 Episode Num: 9 Reward: 486.4106359269747
Total Timesteps: 8354 Episode Num: 10 Reward: 82.01943954108575
Total Timesteps: 8374 Episode Num: 11 Reward: 4.667669991715229
Total Timesteps: 8513 Episode Num: 12 Reward: 53.858602006848365
Total Timesteps: 9167 Episode Num: 13 Reward: 305.0356408144698
Total Timesteps: 10167 Episode Num: 14 Reward: 511.82178935438805
---------------------------------------
Average Reward over the Evaluation Step: 139.579623
---------------------------------------
```

Figure 1. The screenshot of Average Reward obtained by the Agent after exploring on its own without training within 10,000 iterations.



```
---------------------------------------
Total Timesteps: 496323 Episode Num: 526 Reward: 2313.174059019104
Total Timesteps: 497323 Episode Num: 527 Reward: 2304.2822407503345
Total Timesteps: 498323 Episode Num: 528 Reward: 2302.1059113256865
Total Timesteps: 499323 Episode Num: 529 Reward: 2181.903595476909
---------------------------------------
Average Reward over the Evaluation Step: 2271.713480
---------------------------------------
```

Figure 2. The screenshot of the Average Reward obtained by the Agent after the training over 500,000 iterations.



```
----------------------------------------
Total Timesteps: 446323 Episode Num: 476 Reward: 2313.50074836224
Total Timesteps: 447323 Episode Num: 477 Reward: 2268.807594977338
Total Timesteps: 448323 Episode Num: 478 Reward: 2270.8597728399404
Total Timesteps: 449323 Episode Num: 479 Reward: 2348.0461431715325
Total Timesteps: 450323 Episode Num: 480 Reward: 2262.8207243097518
----------------------------------------
Average Reward over the Evaluation Step: 2364.483472
----------------------------------------
```

Figure 3. The screenshot of the Maximum Average Reward obtained by the Agent within 450,323 iterations.

Table 1. Maximum Average Reward over 10 episodes of 500,000 evaluation time steps. Comparing the result of our TD3 model with other state-of-the-art after they experimented their algorithms.

TD3	DDPG	PPO	TRPO	ACKTR	SAC
2364.48	1005.30	1083.20	-75.85	1821.94	655.35

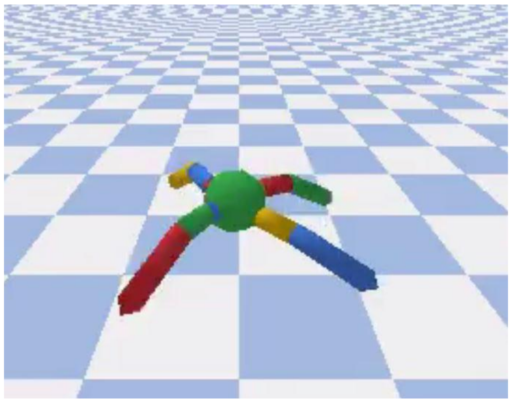


Figure 4. The pybullet environment (AntBulletEnv-v0).

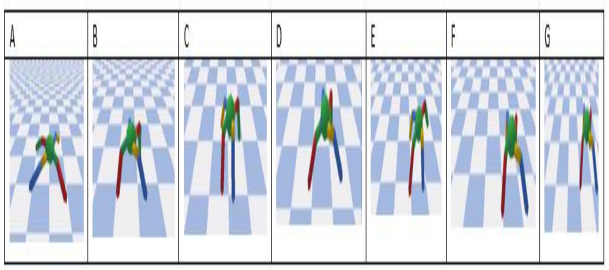


Figure 5. The screenshot of the continuous movement by the 3D Intelligent Ant robot agent from left to right (A-G).

4. CONCLUSION

In this current research work, we modeled and trained an Intelligent 3D 4-Legged-Ant robot to interact with a continuous control environment to walk and run across a field using a current state-of-the-art Twin-Delayed DDPG algorithm. We have elaborated on the evolution process of the TD3 algorithm, as from DQN-DPG-DDPG-TD3. We have evaluated the TD3 algorithm by using the pybullet continuous control task environment and by interfacing it through the OpenAI Gym. Our current TD3 model performed well by achieving a higher Average Reward as compared with other state-of-the-art models such as the DDPG, PPO, TRPO, ACKTR and SAC. We observed that even 500,000 iterations were enough for our model to display such tremendous performance. In future works, this model is applicable to other continuous control tasks such as the Walker2DBulletEnv-v0, HalfCheetahBulletEnv-v0 etc.

Therefore, we can truly state, Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm has prominently improved both the learning speed and performance of the DDPG in a challenging task in a continuous control setting.

ACKNOWLEDGMENTS

First of all, we thank God for his strength and wisdom. Sincere thanks to the Professors and Lab mates in the school of Automation Engineering, University of Science and Technology for their directions. We express our gratitude to the SuperDataScience Team for their technical ideas. we thank the Google Team for providing us with a free virtual machine server for 12 hours. Then lastly, we thank the IASED organizers for given us the opportunity to present our work to the International community.

REFERENCES

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

[2] Erwin Coumans, Yunfei Bai and Jasmine Hsu. PyBullet. Available on: https://pypi.org/project/pybullet/. Retrieved: 5/30/2019

[3] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290, 2018

[4] Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015. Available online: https://arxiv.org/abs/1509.02971

[6] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[7] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems, 2000, pp. 1057–1063.

[8] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In International Conference on Machine Learning, pp. 1889–1897, 2015.

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[10] Scott Fujimoto, Herke van Hoof and David Meger, Addressing Function Approximation Error in Actor-Critic Methods. https://arxiv.org/pdf/1802.09477.pdf, 2018

[11] Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In ICML, 2014.

[12] Sutton, S. Richard and Andrew G. Barto. Reinforcement learning: An introduction. 2nd edition, The MIT Press, Cambridge, Massachusetts, ISBN 9780262039246. Available online: http://www.incompleteideas.net/book/the-book.html, 2018.

[13] Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In AAAI, pp. 2094–2100, 2016.

[14] Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In Advances in Neural Information Processing Systems, pp. 5285–5294, 2017.