Coordinating the Cooperation of Google Research Football Agents via the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) Method

Nicholas C. Park
npark62@gatech.edu

Abstract

Reinforcement learning in a multi-agent setting appears to require additional considerations to the single-agent RL algorithms beyond the independent, per-agent training to account for the complexity that arises from the agents interacting with each other in the environment. Such an interaction can be cooperative, competitive or mixed. Multi-agent deep deterministic policy gradient (MADDPG) is a proposed general-purpose, multi-agent actor-critic learning method that assumes in no way about the environment's structure of agent dynamics and communication channels. Falling under the framework of centralized planning and decentralized execution, this algorithm is examined of its training efficiency and cooperability metrics under a simplified version of a novel RL benchmark called Google Research Football Environment.

1. Introduction

While reinforcement learning has shown success in solving practical problems in the single-agent domain, scaling it to multi-agent systems in a robust way remains a crucial task in modern industrial applications involving multiple intelligent agents. It turns out that training in a multi-agent problem requires further consideration in terms of handling the complexity that emerges from the interactions and partial observability of the agents present in the environment. Also, this type of planning problem can exist in an environment where it is either required or beneficial for the agents to be cooperative, competitive or a mix of both.

Traditional approaches in single-agent reinforcement learning that train each agents independently are ill-equipped to reliably learn the multi-agent coordination (e.g. a cooperation or competition) with regards to the task or environment without additional measures. For instance, policy gradient algorithms applied to each agents independently without the use of any critics are also problematic in cooperative settings, since the group's reward becomes conditioned solely on the each agent's own actions (introducing a high variability). Also, value-based approaches such as DQN or actor-critic methods suffers from the non-stationarity of the environment from any other agent's perspective due to that every agent's policy progressively changes, preventing the use of stabilization via experience replay. [6]

In order to handle above obstacles and to robustly scale RL with no restriction or assumption regarding the cooperative or competitive dynamics of the environment, a multi-agent extension of the deep deterministic policy gradient algorithm, MADDPG, is introduced under the framework of *centralized planning and decentralized execution* of the policies.

We hypothesize that this specific approach of the multi-agent actor critic method will increase the cooperative behavior between the agents in order to aid the overall ability of the "team" to achieve rewards. As a test, this algorithm is implemented in the Google Research Football Environment, a novel multi-agent RL benchmark, to train the policies of homogeneous agents to maximize the average reward, achieved by the scoring of goals in the football gameplay.

The performance of the algorithm is evaluated in the win rate improvement over three other baseline teams, all trained by the Proximal Policy Optimization (PPO) algorithm that vary in hyperparameter selection or model architecture. Additionally, two other manually collected metrics, pass success rate and shoot-to-pass ratio, are examined over the episodes to measure the degree of teamwork and successful cooperation of the trained team in comparison to the baseline teams.

2. Game Environment and the Goal

Google Research Football Environment is a novel reinforcement learning environment with benchmark problems for RL research, where the main goal is to beat the hand-engineered and rule-based "AI" teams with varying difficulties [3]. This is representative of the multi-agent extension of the MDP, formally described as a Markov game [5]. The tested, particular environment setting in this paper reduces the benchmark to involve only 3 players in each team (i.e., 3 vs. 3) resulting in a Markov game of just two agents, since the team's goalkeeper and the entirety of the opponent team operates based on the hand-engineered, rule-based "AI bot".

This environment's action space is the product of the action spaces of the two agents, each of which is a 19-dimensional, discrete vector that realizes the player's possible movements and ball-handling. Among a few state representation available from Google Research Football, the one used here is a 43-dimensional, continuous observation space vector that describes the global, positional and directional configuration of the 3+3 players in the game and assumes no partial observability between the agents.

The reward function for each agent is identically +1 if the left team (i.e. the training) scores and -1 if the right team (i.e. the opponent) scores, and 0 otherwise. We additionally introduce a shaping of the rewards beyond this, only during training.

To motivate the agents to bring the ball closer to the goal, they receive a non-reversible, non-repeatable reward of +0.1 for each of the 9 equally spaced checkpoints crossed between the field-center and the goal. Further, to prevent the agent from learning to only cross the checkpoints without actually scoring, the reward for the uncrossed checkpoints is also given if the agent score at any location in the field (i.e., a reward of +1.9 total is given when a goal is scored). An episode terminates when a team scores, the ball goes out of the bounds, or its horizon of 500 steps is reached.

The goal in this multi-agent environment is to learn a decentralized, deterministic set of policies of the agents that maximizes the performance in expected rewards by obtaining the best training model checkpoint in terms of the metric: mean episode reward.

3. Approach & Algorithm

The multi-agent deep deterministic policy gradient algorithms (MADDPG) are the multi-agent extension of the deep deterministic policy gradient algorithms (DDPG) that allow centralized planning and decentralized execution. Moreover, DDPG is a special case of deterministic policy gradients (DPG) that uses deep neural networks, while DPG is the limiting case of the stochastic policy gradient (PG) as its variance tends to zero. [1]

3.1. Background: DPG & DDPG

PG algorithms have shown success in solving continuous action space problems. Stochastic policies can directly represent the policy using a parametric probability distribution over actions given a state. In the case of discrete actions, policy improvement by globally maximizing over the estimated action values can be done easily, such as in Q-Learning. However, greedy policy improvement at each step by globally maximizing is not tractable in the continuous case. Instead, the PG algorithms iteratively improve the policy to maximize the cumulative reward from following this policy by ascending its gradient with respect to the policy's parameters.

DPG algorithms set the maximized objective of PG to be $J(\mu_\theta) = \mathbb{E}_{s\sim\mathcal{D}}[r(s, \mu_\theta(s))]$, or the expectation of the action value of following a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$ over the visited states where $\mathcal{D}$ is the state distribution. We can then use the sample mean of the gradient to update the policy at step k as the following [1]:

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))\right] \tag{1}$$

With chain rule we can decompose the gradient into the so-called actors and critics:

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu^k}(s, a)\Big|_{a=\mu_\theta(s)}\right] \tag{2}$$

where the μ serves as the actor and, Q^μ, the critic. Similar to DQN's adoption of Q-Learning, DDPG is a off-policy variant of DPG which introduces deep neural networks to approximate its μ and Q^μ, as well as the stabilization of learning by sampling experiences from a replay buffer and delayed-target-network-based updates. [7].

3.2. Centralized Planning & Decentralized Execution

MADDPG seeks to reduce the variance with the multi-agent extension of the above deterministic actor-critic gradient update procedure to the N policies $\mu = \{\mu_1, ..., \mu_N\}$, where each agent i seeks to maximize its own rewards (which can often be jointly shared in a cooperative setting). However, when learning each agent's policy independently, the variance tends to surge. The non-stationary environment prevents the correct use of experience replay when evaluating actions, and the reward also becomes inadequately conditioned by the agents' own actions. [6]

3.3. MADDPG

To work around this, MADDPG allows each policy to learn under the guidance of a centralized critic with a global set of observations of all the agents during training (i.e., centralized planning). Specifically, a critic is augmented with extra information into a centralized action-value function $Q^{\mu_i}(o_i, ..., o_N, a_1, ..., a_N, ...)$ to guide its actor's gradient updates. The purpose is for each policy to be able to pick an action $\mu_i(o_i|\theta^{\mu_i})$ solely based on its private observation o_i during testing (i.e., decentralized execution). The gradient of $J(\mu_i)$, agent i's expected rewards, with respect to the policy weights of agent $i, \forall i \in [1...N]$ becomes:

$$\nabla_{\theta_i} J(\mu_i|\theta_i) = \mathbb{E}_{X,A\sim\mathcal{D}}\left[\nabla_{\theta_i}\mu_i(o_i)\nabla_{a_i}Q^{\mu_i}(X, A)\right] \tag{3}$$

where $o_j \in X, \forall j \in [1...N]$ & $A = [\mu_1(o_1), ..., \mu_N(o_N)]$ in Q^{μ_i} above, and $\mathcal{D}$ denotes a replay buffer of experience tuples of the form: $X, A, r_i, ..., r_N, X'$. Note the global state X in the Q^{μ_i} can add more information other than the agents' observation to learn mixed cooperative-competitive behavior, such as to include conflicting rewards, which makes this algorithm more robust to various environment dynamics. [6]

For every agent, as in DDPG, a delayed target actor network μ_i' is maintained to update its corresponding centralized critic in the direction of minimizing the MSE loss below:

$$\mathcal{L}(\theta_i) = \mathbb{E}[(r_i + \gamma Q^{\mu_i'}(X', A') - Q^{\mu_i}(X, A))^2] \tag{4}$$

where $o_j' \in X', \forall j \in [1...N]$, $A' = [\mu_1'(o_1'), ..., \mu_N'(o_N')]$. Further, to avoid assuming the knowledge of other agent's policy μ_i' during this critic update, each agent i can instead learn an approximate target policy $\hat{\mu}_i'^j$ for each of the other $N-1$ agents, and instead can use $A_i' = [\hat{\mu}_i'^1(o_1'), ..., \hat{\mu}_i'(o_i'), ..., \hat{\mu}_i'^N(o_N')]$ to compute the loss. More tricks to reduce the variance are in disposal such as ensembling the policies. (Full algorithm outlined in Lowe et al.) [6]

4. Implementation

The open-source python library for reinforcement learning called RLlib is used to implement, train and tune the MADDPG agent. This implementation allows for per-episode, custom callbacks of various metrics with respect to the episode of the environment (i.e. Google Research Football Environment) during model training, from which the best model checkpoints in terms of the mean episode reward can be retrieved. Here the multi-agent setting is implemented by registering the environment with a grouping of the agents present and specifying its observation and action space with the tuples of each agent's observation and action spaces. The policies are maintained grouped, with experience replay done together in a ”lockstep” mode, but are separately called to sample actions based on their observation during execution.

4.1. Tuning and Training

Each of the MADDPG-specific hyperparameter is tuned by sampling it from a specified range based on a chosen continuous distribution or discrete choices to create one sample run. Several samples of MADDPG are run in parallel for 2 million timesteps to optimize over the metric: mean score reward. The list of actively tuned hyperparameters is as follows: actor_hiddens = [128,62], actor_lr = 0.00335927, actor_feature_reg = 0.005, actor_activation = relu, critic_hiddens = [256,256], critic_lr = 0.00818687, critic_activation = relu, train_batch_size = 32, gamma = 0.99899474, n_step = 5.

A good initial sampling range for a hyperparameter is not known prior to some experimentation, so the tuning of a hyperparameter was first done using four sparse discrete choices, with all else kept equal. This was repeated with another four narrower discrete choices of the hyperparameter in the vicinity of the best of the previous values. Once every hyperparameter has sufficiently narrowed, a grid search sampling is done over the resulting hyperparameter space for 1 million steps each to ensure the best selection, with an ASHA scheduler to induce an aggressive early-stopping [4].

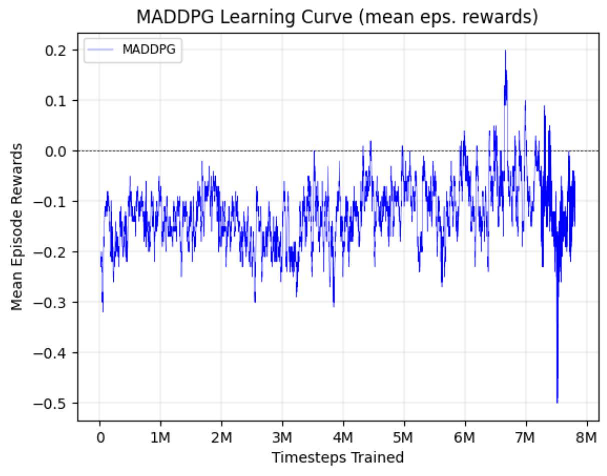

Figure 1. Smoothed mean episode reward of MADDPG training

The finally tuned model is trained independently for 8 million timesteps. In Figure 1, the mean episode reward of the MADDPG agent is smoothed by a 15-timestep moving average. The mean here indicates the average from the batch of episode stats from the most recent training iteration. A steady uptrend is observed with a peak mean episode score reward = 0.2 at timestep = 6.684M, at which the checkpoint for this model is retrieved. Various statistics are computed and called back to observe the training with respect to variables other than the mean score rewards.

One of the key goals in this experiment is to see the MADDPG's trainability of cooperation between the players which we evaluate in section 5. It is interesting to note that during training, the episode mean value of the number of successful passes peaks at 0.54 (i.e., more than double the total mean of this value, 0.252892) at the 6.6 millionth timestep, very adjacent to where the best checkpoint is achieved.

4.2. Exploration

The typical method of exploration in the DDPG method variants is to introduce a random noise process to the policy, such as the Ornstein-Uhlenbeck noise [7]. However, we avoid using the noise and take advantage of the RLLib implementation of the policy whose output is returned as a Gumbel-Softmax output of the action logits. (The algorithm is implemented in RLlib for the continuous action space with a modification to allow discrete action selection.) With this softmax action selection, we can appropriately facilitate the exploration of all 19 actions with non-zero probabilities. Also, its softness can be regulated by annealing the temperature. This is advantageous when the ”next best action” in terms of the probability distribution falls within a cluster of similar actions. For example, in Google Football, the actions 1 through 8 displaces the player, while actions 9 through 12 deals with kicking the ball for passing/shooting, and so forth.

4.3. Evaluating

During the testing phase, the softmax action selection is cooled to the limit to become an argmax (i.e., a greedy action selection). The gradient updates are turned off, and episodes are run by directly computing the single action of the agent's trained policy per player. This is repeated following the environment's response step according to the input of the agent's action. This loop repeats until the episode terminates. This is done whilst computing several episodal metrics along with the game results (win/tie/loss). Specifically, the counts of pass attempts, shoot attempts and successful passes of the gameplay are obtained. Statistics regarding these like the mean, variance, min, max, etc. are computed over 100 episodes to observe the performance and behavior of the trained agent.

5. Results and Analysis

In this section, the trained algorithms are compared of the metrics: win rate, pass success rate and shoot-to-pass ratio. We first formally introduce the three pre-trained baselines to compare the MADDPG's performance against. These baselines are the best checkpoints of the mean episode score reward of the proximal policy optimization (PPO) trained agents, a family of PG methods that utilizes a novel, ”surrogate” objective function to enable multiple epochs of minibatch update with good sample complexity [2]. The three PPO baselines are different in the tuned hyperparameter values or their architectures (e.g. relu vs. tanh activation of fully connected layers, the presence of LSTM in the model, etc.).

Figure 2 shows the win rate improvement over the training of MADDPG and PPO using the final, tuned models for 8 million timesteps. We first note the training efficiency of MADDPG over the PPO baselines with respect to the mean win rate metric. A more rapid learning of the MADDPG agent can be observed from the relatively frequent peaking of the win rate such as in the timesteps 3.2M to 3.6M, 4.2M to 4.5M and 6.5M to 7M.

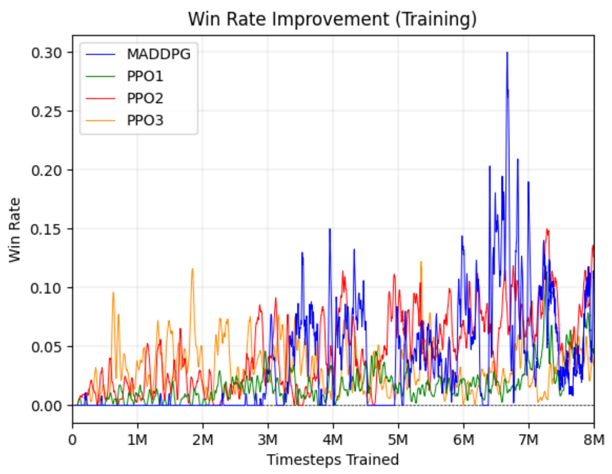

Figure 2. Mean win rate measured during training

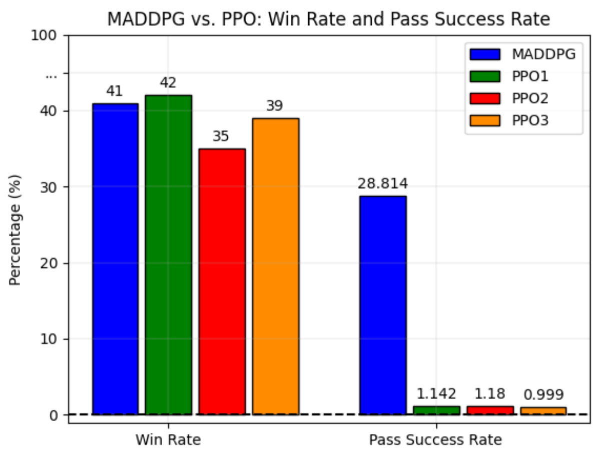

Figure 3. Comparison of win & pass success rates over 100 episodes

The mean win rates at these timesteps appear to correlate with the mean score rewards shown in Figure 1.

However, it is crucial to note that only the MADDPG checkpoint is trained for 8 million time steps, while each of the three PPO baselines are trained for 50 million steps, to showcase the training efficiency of MADDPG. Since MADDPG was trained for less than 20% of the timesteps used to trained the PPO baselines, if MADDPG can perform similar to or better than the PPO baselines, this algorithm can be seen as a promising candidate to the multi-agent football benchmark which is purported to entail a more complex and hierarchical decision making than the SMAC benchmark.

5.1. Win Rate: MADDPG vs. PPO

We observe that even with less than 20% of the trained timesteps of the PPO, the win rate of the MADDPG-trained agents is similar to the PPO baselines over 100 evaluation episodes. The left side of Figure 3 shows the bar graph comparison of the number of wins out of the 100 episodes.

This is particularly striking in that the MADDPG model did not use any recurrent neural networks like LSTM such as in PPO baselines 1 and 3 to pre-process the agent's observation. MADDPG observed here simply uses 1 fully connected layer for each of the actor and critic and does not involve any pre-processing layers. This highlights that MADDPG's lightweight version has the ability to achieve similar levels of performance compared to the PPO agents that are heavier and trained for 6 times longer in timesteps. As what contributes to the high win rate of MADDPG, the pass success rates over these gameplay episodes are examined.

5.2. Pass Success Rate: MADDPG vs. PPO

First, it is no surprise that good passing in football provides a higher chance of scoring the goal. It turns out that MADDPG achieves exactly this ability to cooperate by passing the ball well from one player to another, while PPO fails to achieve this. The **pass success rate** of an episode is derived from calling back two manually computed metrics: 1) the number of **successful passes**, (divided by the) 2) number of **pass attempts**. **Pass attempts** is the total count of the actions 9, 10 and 11 (long, high and short passes, respectively) within an episode. **Successful passes** is the number of times the ball owned player successively changes either from player 1 to 2 or from player 2 to 1 due to the pass. However, any passes involving player 0 (the goalkeeper) are ignored as it is not a controlled part of the agent to be considered a cooperation. This success rate is averaged over the evaluation episodes.

From the right side of Figure 3, it is shown the pass success rate of MADDPG significantly outperforms the PPO baselines. The ability to score goals and win by the method of cooperating appears specific to the MADDPG agents. We can further examine the underlying metrics.

Interestingly, despite MADDPG's high pass rate relative to the PPO's, the number of successful passes is more or less similar between the algorithms over the course of all the actions taken in the evaluated episodes. What differs turns out to be the number of total pass attempts, which is much smaller in MADDPG as shown in Figure 4. This suggests that MADDPG makes a carefully coordinated pass instead of haphazardly and abundantly outputting pass actions from the policy.

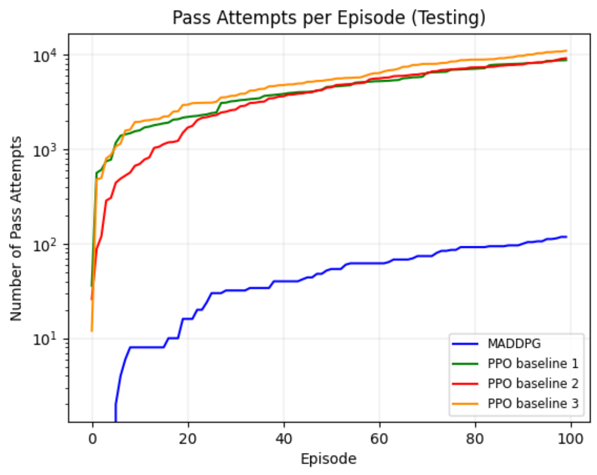

Figure 4. MADDPG does not act random passes. (Log scale y-axis)

This makes us wonder, how then are PPO baselines able to achieve similar levels of mean win rates or score rewards? The answer to this turns out to be explained by another manually computed metric: shoot-to-pass ratio.

5.3. Shoot-to-Pass Ratio: MADDPG vs. PPO

The **shoot-to-pass ratio** is derived by another callback, **shoot attempts**, denoting the total number of the action 12 within an episode that performs a shot in the direction of the opponent's goal. This metric is divided by **successful passes** to obtain the ratio.

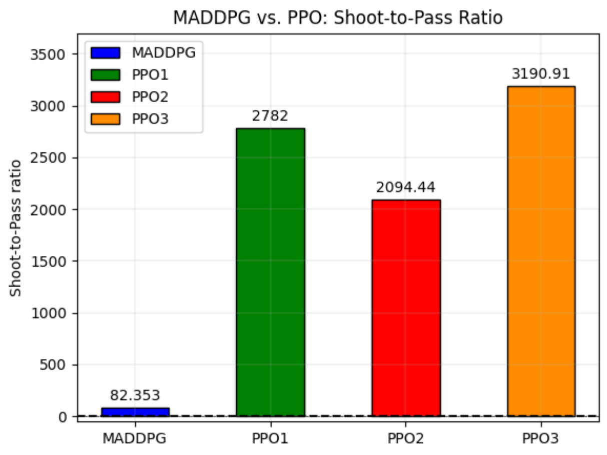

Figure 5. Comparison of Shoot-to-pass Ratio over 100 episodes

The bar graph comparison of this ratio in Figure 5 shows that PPO takes the shot towards the opponent's goal more often than MADDPG does and appears to be particularly well-trained at optimizing the success probability of this action. This is supported by Figure 6, where the cumulative sums of the number of shoot attempts are examined in comparison over the 100 episodes.

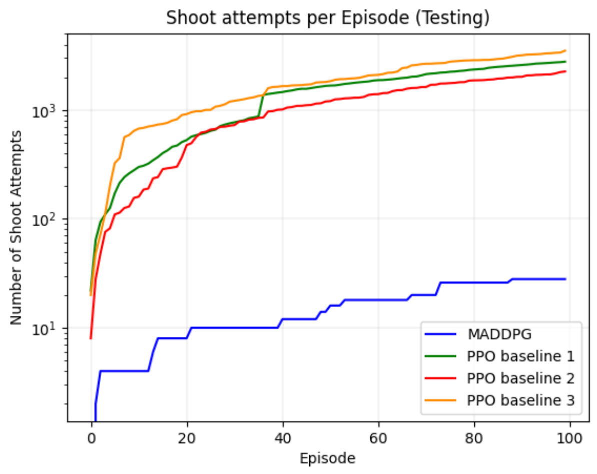

Figure 6. MADDPG does not take random shots towards the goal

6. Remarks

The experiment results reveal a contrast between the trained behaviors of MADDPG and PPO algorithms in this environment, specifically, the ability to cooperate vs. to make good shots. It appears that MADDPG and PPO baselines are trained to optimize the success chance of different strategies to effectively score higher, that is, the pass success rate and the shoot success rate, respectively. We observe that not only does MADDPG outperform the baseline agents in terms of training efficiency, it also provides a cooperative coordination of the agents in the Google Research Football environment.

7. Pitfalls & Areas of Improvement

If the time allowed, more focus would be spent on debugging the "contrib/MADDPG" algorithm class of RLLib to enable the pre-processing of the states via a deeper neural network model including recurrent layers, which the configuration documentation of the algorithm class claims to allow. In this case, the error occurred due to a deprecated attribute "get_model" of the algorithm-agnostic class ModelCatalog which now has become get_model_v2 and replacing this to enable the state pre-processing did not have a trivial workaround. Enhancing the policy with a more complex form of neural network could have led an interesting ablation study.

The supported deep learning framework for MADDPG was limited to Tensorflow, which unlike PyTorch required a driver-dependency troubleshooting in order to employ the GPU on the Linux docker for the task. The algorithm required matching the supported version of Tensorflow's probabilistic model handling library, the parallel computing application toolkit (CUDA) and importing this toolkit's native deep neural network library to enable the parallelism mentioned above.

References

[1] Nicolas Heess Thomas Degris Daan Wierstra Martin Riedmiller David Silver, Guy Lever. Deterministic policy gradient algorithms. http://proceedings.mlr.press/v32/silver14.pdf. 2

[2] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. https://arxiv.org/abs/1707.06347. 3

[3] Karol Kurach and Olivier Bachem. Introducing google research football: A novel reinforcement learning environment. https://ai.googleblog.com/2019/06/introducing-google-research-football.html. 1

[4] Afshin Rostamizadeh Ekaterina Gonina Moritz Hardt Benjamin Recht Ameet Talwalkar Liam Li, Kevin Jamieson. A system for massively parallel hyperparameter tuning. 3

[5] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994. 1

[6] Aviv Tamar Jean Harb Pieter Abbeel Igor Mordatch Ryan Lowe, Yi Wu. Multi-agent actor-critic for mixed cooperative-competitive environments. Mar 2020. https://arxiv.org/abs/1706.02275. 1, 2

[7] Alexander Pritzel Nicolas Heess Tom Erez Yuval Tassa David Silver Daan Wierstra Timothy P. Lillicrap, Jonathan J. Hunt. Continuous control with deep reinforcement learning. page 5, Sept. 2015. https://arxiv.org/abs/1509.02971. 2, 3

$ *git log -1 –format="%H"*
>> 47a8426ce32d9f702e7b73e4cc7b82e3a521d73b

Table 1. Latest Git Commit Hash