Parametrized Control in Soccer Simulation with Deep Reinforcement Learning

Yang Xu[1,a)] Tsuruoka Yoshimasa[2]

Abstract: The recent advances in deep reinforcement learning have proved itself as a very promising and competitive solution for making sequential decision in games with discrete action space and well-defined reward signal, e.g., Atari games [3] and robotics simulation[11]. However, when applied outside of these experimental environments deep reinforcement learning is still challenged by increase of state and action space dimensions. Also, when dealing with more complex environments with continuous or parametrized action space. Deep reinforcement learning does not always display stable learning. In this work we introduce Half Field Offense, a new experimental environment for deep reinforcement learning. We tested some state-of-the-art algorithms in this new environment to test their adaptivity and performance.

1. Introduction

One very crucial and important goal of artificial intelligence is to allow artificial intelligence controlled agents to learn and carry out real-life tasks. In the recent development of deep reinforcement learning a lot of research is focused on using computer game environments to develop algorithms that have potential to be applied in real-life tasks. Many successes in the game environments have shown promising future for deep reinforcement learning. The "Deep Q Network" [3] have successfully allowed a deep reinforcement learning agent to learn to play almost all different Atari-2600 games with near human performance. In 2015, Schulman et al. was able to let an agent learn to complete several tasks of robotics simulation in the MuJoCo [11] environment. Furthermore, a deep reinforcement agent powered by Deep Deterministic Policy Gradient (DDPG) is able to learn to the same set of tasks with less training time and similar performance [4].

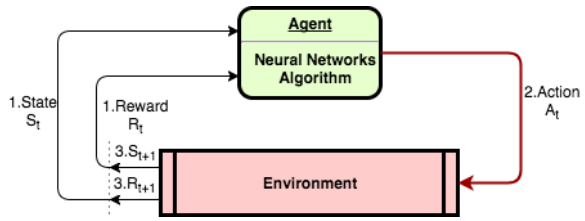


Fig. 1: Basic mechanism of deep reinforcement learning.

Basic mechanism of deep reinforcement learning is displayed in Fig.1. Where an agent observes the state of an environment and then issue an action according to the agent's policy. This policy is produced by the interpretation of the agent's deep neural networks. The environment receives the action and proceeds into the next state while providing a reward signal to the agent. This reward signal serves as a reference for the network to update itself.

More formally, we consider the games played by the deep reinforcement learning agent as a Partially Observable Markov Decision Process (POMDP) defined by the tuple $(S, A, P, r, \mu_0, \gamma)$. S is a set of states, A is a set of actions, $P : SxAxS \rightarrow [0, 1]$ is the transition probability, $r : SxA \rightarrow \mathbb{R}$ is the reward function, μ_0 is the initial state distribution and γ is the discount factor. These elements of reinforcement learning will be described in detail in Chapter 2.

Most of these previous successes are achieved in experimental environments such as OpenAI's gym-Atari [12] or MuJoCo [11]. There are a some points making tasks in these environments different from actual real-life tasks. Firstly, the action space of these tasks are relatively small. The Atari tasks contains only 9 discrete actions. Whereas the MuJoCo tasks has an action space of smaller than 6 continuous actions. In Real-life it is very easy to find tasks with more than 20 available actions. Secondly, these experimental environments has a relatively small size of state space. For Atari tasks, simplified game images are used to represent states. which are represented as a 80 x 80 array of discrete numbers. For the MuJoCo tasks, feature sets of various length (normally smaller than 30 continuously figured features per feature set) are used to represent the state. The size of features or state representations in real world is much larger. Thirdly, these environments provide a true reward for the tasks the agent is carrying out. This is not always true for real-life tasks as we normally can not provide a true reward. Typical real-life tasks like auto driving cannot be easily evaluated to provide exact figure for reward signals.

[1] Dept. of Electrical Engineering and Information Systems, Graduate School of Engineering, the University of Tokyo.
[2] Dept. of Information and Communication Engineering, Faculty of Engineering, the University of Tokyo.
[a)] yangxu@logos.t.u-tokyo.ac.jp

Agent developers would need to provide an approximation to the reward by either hard code a reward function or use a reward approximator.

Another problem of the state-of-the-art deep reinforcement learning achievements is the lack of ability to learn hierarchical tasks. This problem have first started to emerge from the original DQN work. In the DQN work the agent scored very low in a game called Montezuma's Revenge. In order to get high score in this game, an agent has to do logical and rational moves like gather a key to open a door. However the DQN agent failed to understand this simple logic. This problem continues to trouble researchers in later works. Methods like apply human language as a guide [16] or complex actor-critic structure [15] have been proposed. But most of them still focus on the Montezuma's revenge game and are not generalized into other environments.

In order to solve the above issues on the current testing environments for deep reinforcement learning. We need to introduce a more complex and more close-to-reality testing environment. In this work we used the Half Field Offense (HFO) [1]. This platform origins from the RoboCup 2D competition platform, which is a testing and competition platform with long history. HFO provides possibilities for researchers to explore single agent, multi-agent and teamwork tasks for learning agents. The HFO environment provides a large continuous low-level state space and a parametrized continuous action space. More specifically, this action space requires the agent to select both the type of the action and the parameters of the action. This structure of action selection is very commonly seen in real life but have not been found in other testing environments. HFO's tasks are all hierarchical. Even the simplest task we carried out in this work involves multiple steps for the agent to learn if high reward is desired.

In this work we apply some state-of-the-art deep reinforcement learning algorithms in HFO. We test the performance and adaptivity of these algorithms by letting the agents learn to complete a simple task in the HFO domain with minimum tuning of hyper parameters and neural network structures. Despite difference in learning pace and style. All of the tested algorithms have shown stable learning behaviors towards positive rewards.

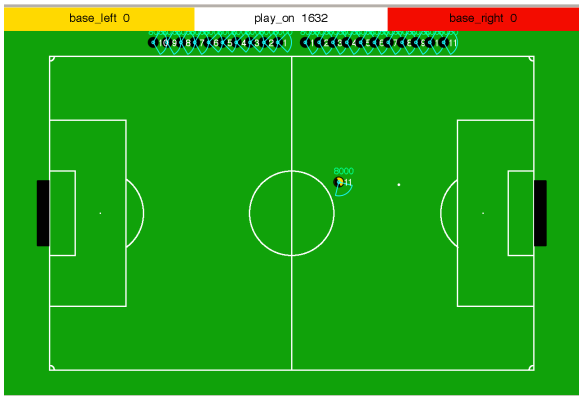
The rest of the papers will first introduce the characteristics of the Half Field Offense environment. And then describe the methods used in this work. Followed by the information on experiment settings and experiment results.

2. The Half Field Offense Environment

The Half Field Offense (HFO) environment origins from RoboCup 2D, a 2D soccer simulation league belongs to the RoboCup international competition that promotes Artificial Intelligence and robotics development. It was then coded into a reinforcement learning algorithm by Hausknecht and Stone. The environment abstracts a whole soccer game into a two dimensional representation. HFO allows researchers to carry out full game of soccer as well as soccer related mini tasks.

The actual program of the environment is composed by a game server and agent clients. With each agent client run as an individual process on the host PC. The HFO environment can generally be modeled as a Partially Observable Markov Decision Process (POMDP) as each agent can only obtain limited information about other players in the game. However, in the specific task we carry out in this work. Only one player is in the game. Therefore the game can be considered as a Markov Decision Process.

In this work we train agents based on different algorithms to learn to score a goal against no defenders. This can be considered as the most easy task in HFO. Details of experiment setting are shown in Chapter 4. Details about the task is discussed in the next sub-section.



A screen-shot of an agent trying to score against no defenders in HFO.

2.1 HFO Task Design

In order to show the performance of existing algorithms we let our agent to learn to play a simple task in HFO. The agent needs to score a goal with no other players defending. Start conditions are the following:

- The player is spawn on a random spot on the pitch
- The ball is placed on a random spot near the center circle of the pitch.
- The ultimate goal for the player is to put the ball into the goal area, which is located on the center rightmost side of the pitch.

In order to finish this task fast and stably. We have handcrafted a reward function to encourage relative behaviors in different state of the game. The details of the reward function can be found in Table 3 and chapter 2.4. The desired behavior we want to agent to learn is:

- Run towards the ball when the game starts.
- After getting control of the ball, the agent should make the ball closer to the goal by kicking it with appropriate power and angle.
- Eventually the ball would be kicked into the goal or out of bound. If none of these happed. The game finishes itself after 100 time-steps.

While being seemingly simple, this task is hierarchical. As

we mentioned in the previous chapter, hierarchical tasks are considered relatively hard for the existing algorithms to learn. Also from the state and action space descriptions below we can find that they are higher than the existing tasks in Atari and MuJoCo.

2.2 State Space

Feature Type	Example feature	Example Value
Landmark	Goal Centre	(23.1, 32.1, 50)
Angle	Self_Ang	-180.0
Valid	Self_Pos_Valid	1.0
Proximity	Ball_Dist	0.35
Other	Stamina	8000
Boolean	Colliding_with_ball	0.0

Table 1: Categories of features and example features in HFO

The HFO environment can provide three different levels of feature sets that abstracts different level of information. In this work we use the low level feature set as we aim to explore efficiency of algorithms in large state space. The state set contains 58 continuously-valued features. Features are composed by information including agent's own stamina and position, the distance and angle to the objects on the pitch, whether the agent is having control of the ball and whether the agent is able to kick the ball. These features are categorized into several types. Examples of categories and features are shown in Table 1. The full list of states can be found in the documentation of HFO [8].

Compared to the MuJoCo environment which also uses continuous feature array with maximum size of 20 (depending on the actual robot simulation task that the agent is carrying out) to represent the state of the game, HFO has a much larger state space. This makes the training time much longer than MuJoCo tasks. This is stated in the Results section.

2.3 Action Space

Action	Parameters	Min Value	Max Value
Dash	Power	0.0	100.0
	Direction	-180.0	180.0
Kick	Power	0.0	100.0
	Direction	-180.0	180.0
Turn	Direction	-180.0	180.0
Tackle	Direction	-180.0	180.0
Catch	Direction	-180.0	180.0
None	N/A	N/A	N/A

Table 2: Actions and relevant parameters

In each time-step of HFO, the agent needs to select an action and a parameter set related to the action. All the available actions and related parameters are listed in Table 2. In this work our agent only needs to play as an attacker. Therefore, the defender specific *catch* and *tackle* actions are not being considered for our agents.

As stated in the previous section, the HFO environment holds a very special characteristic. For every action one or more relevant parameters are required. This is a new challenge for the existing deep reinforcement learning algorithms. The capability of these algorithms to relate and understand the relationship between the action and its parameters.

2.4 Reward Function

One very important perspective of the reinforcement learning scheme is the reward signal. As we mentioned above, environments like Atari provides true reward to the agent during playing and training. This makes the accuracy of reward signal very high and allow the agent to learn and complete tasks easily. However, in real-life tasks developers have to figure out the reward function on their own. The same problem exists in HFO. In this work we have little intention on designing an accurate and scientific reward function. We use a rough approximation for the reward, which was provided by Hausknecht and Stone [2] .

The reward is composed by four major parts. The *move* reward, the *kick* reward, the *ball-controlling* reward, and the *goal* reward. Detailed values and definition of rewards are shown in Table 3. Since the coordinate data was normalized to be between -1 and 1 in the HFO environment. The *move*, *ball-controlling* and *kick* reward will be much smaller than the *goal* reward. Therefore, scoring behaviors are emphasized and strongly encouraged.

2.5 Related Works

The initiative of this work is inspired by Duan et al. [7]. They have benchmarked some state-of-the-art deep reinforcement learning algorithms using similar settings and training time to benchmark existing deep reinforcement learning algorithms on tasks in Atari and MuJoCo. Their work provides great view on the existing algorithms. But it is still constrained in these experimental environments.

Besides [7], We have also observed that even reproducing results of previous work in the same environment is becoming a serious problem for a lot deep reinforcement learning researchers. This further proves that some extended test of existing algorithms are required to show the capability of them adapting to different environments under different hyper parameters.

Hausknecht and Stone have successfully implemented DDPG in the HFO environment [2]. Which provides a strong argument on the effectiveness of existing deep reinforcement learning in HFO. Other work on the HFO environment are mainly focused on doing multi-agent tasks with relatively naive reinforcement algorithms (non-deep learning based) [9][10].

3. Methods involved

To select potentially good algorithm for the HFO environment given that we have limited time is a big challenge. In the end we designed to apply three methods. The first one is Deep Deterministic Policy Gradient (DDPG) method. We decided to use this not only because the previous work using DDPG by Hausknecht and Stone [2] have proved its efficiency and effectiveness, but also because DDPG showed

Reward	Value	Condition
move	Distance of agent moved towards the ball	Awarded when agent moves. This may be negative if the agent move away from the ball.
kick	Distance of ball moved towards the goal	Awarded when ball is kicked. This may be negative if the ball is kicked away from the goal.
ball-controlling	2	Awarded when agent first gain control of the ball
goal	5	Awarded when the game ends in goal status

Table 3: Rewards and conditions

great performance in the work by Duan et al. [7]. Moreover, DDPG is a classical and representing algorithm for on-line policy update algorithm.

Based on Duan et al.'s benchmarking of algorithms, NPO based methods such as TRPO and TNPG was the top performer of most of the tasks. This makes these two algorithms very attractive. Besides, these methods have never been applied to the HFO environment before. Therefore, these 2 algorithms are used in our experiments as well.

Besides these we have also looked at other classical approaches in reinforcement learning. Such as Cross Entropy Method (CEM) [13] and Covariance Matrix Adaption Evolution Strategy (CEA-ES) [14]. However these methods does not out-stands in previous reports. We therefore putting the testing of these in the future plan.

Deep Deterministic Policy Gradient

The Deep Deterministic Policy Gradient algorithm was first proposed by Lillicrap et al. [4] as a successor inspired by Deep-Q Network (DQN) [3]. Unlike the DQN which is designed to play games with a discrete action space, DDPG is capable of playing games with a continuous action space. DDPG utilizes two deep neural networks to complete the task. A policy network is used to generate a policy and a critic network is used to judge this policy and provides the policy network the gradients for parameter update. This critic network updates itself by using the reward signal obtained during game play. Updates of both networks are done by sampling game play trajectories randomly. In order to adapt DDPG to the HFO environment, some tricks like bounding action parameters by adjusting the update gradients of neural networks are introduced.

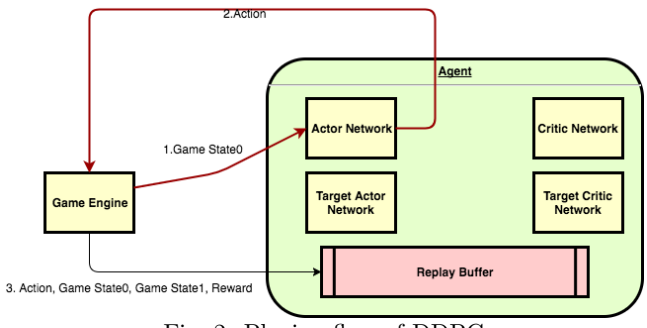


Fig. 2: Playing flow of DDPG

We now explains the mechanism of DDPG. If we define state as s, action as a, and an action selected at time step t as a_t. We have our policy network monitored as $\mu(s)$ and critic network defined as $Q(s, a)$

As displayed in Fig. 3, the Action selection process when playing the game for DDPG is fairly simple. The algorithm selects an action by:

$$a_t = \mu(s_t|\theta^\mu) \tag{1}$$

where θ^μ stands for the parameters in the actor neural network.

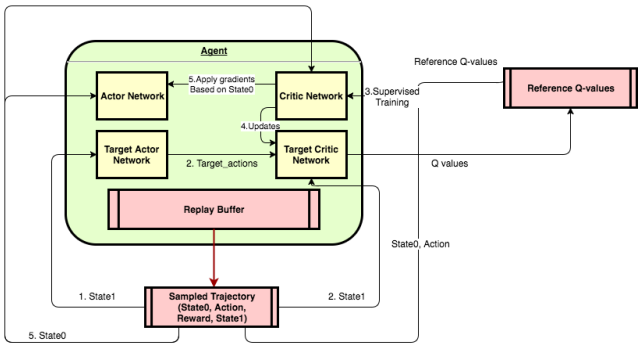


Fig. 3: Training flow of DDPG

The training and updating of DDPG's two neural network based function approximators is a bit complicated. The update of critic is done by minimizing the loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i \mid \theta^Q))^2 \tag{2}$$

Where y_i is defined as:

$$y_i = reward_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1} \mid \theta^{\mu^{'}} \mid \theta^{Q^{'}}) \tag{3}$$

Whereas the policy network was updated by calculating the gradient of action output of policy network against the Q value output of the critic network.

$$\Delta_{\theta^\mu} J \approx \frac{1}{N}\sum_i \Delta_a Q(s, a|\theta^Q) \mid_{s=s_i, a=\mu(s_i)} \Delta_{\theta^\mu}\mu(s \mid \theta^\mu) \mid s_i \tag{4}$$

Natural Policy Gradient (NPO) Methods

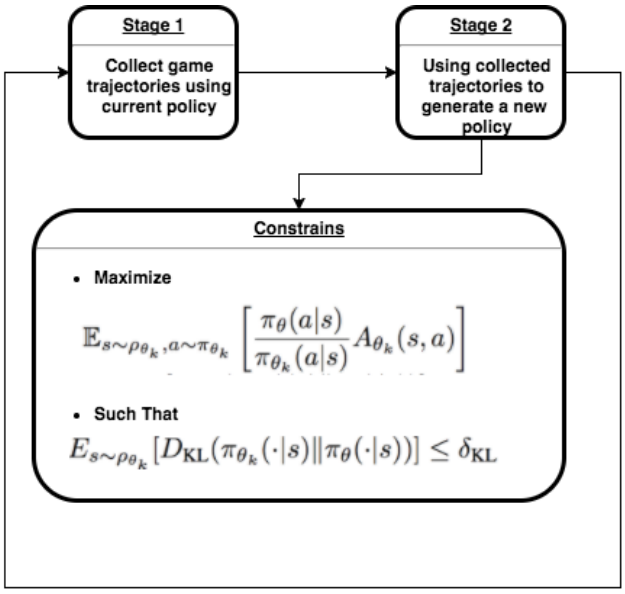


Fig. 4: Flow of TRPO algorithm

The idea of NPO was first proposed in 2002 by S. Kakade [6] and the best example working implementation of it is Trust Region Policy Optimization (TRPO) [5]. Instead of using critic gradients to update the policy, which may lead to unstable updates and a worse policy. TRPO aims to always find a better policy and always let the average reward increase. After collecting trajectories, the algorithm updates its current policy network by calculating an advantage value. This advantage value represents the reward difference between a new policy and the old policy. A series of techniques were introduced in the TRPO algorithm to reduce the amount of calculation for the advantage value. In theory this method always produces a better policy. We also applied another version of the NPO algorithm, namely Truncated Natural Policy Gradient (TNPG). This algorithm is similar in mechanism to TRPO but it only backtracks once when optimizing the parameters. It has been proved that it has good performance in both Atari and MuJoCo tasks [7]. However, it demonstrated similar problems of increasing perplexity during training like the TRPO algorithm.

The induction steps of TRPO and TNPG are rather complicated [5]. The final algorithm flow is displayed in Fig.5. Where the algorithm maximizes θ in:

$$\mathbb{E}_{s\sim\rho_{\theta_{old}},a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)}A_{\theta_{old}}(s,a)\right] \tag{5}$$

subject to:

$$\mathbb{E}_{s\sim\rho_{\theta_{old}}}\left[D_{KL}(\pi_{\theta_{old}}(\cdot\mid s)\parallel\pi_\theta(\cdot\mid s))\right]\leq\delta. \tag{6}$$

4. Experiments

In the design of experiments we followed some previous success experiments' hyper parameter settings. Some rational adaptations are also applied during the experiments. Some of them have shown significant improvements on the performance of agents. General design principles are listed as below:

- Due to the limitation of time we are not able to training every agent till convergence. Therefore we train each agent for a fixed time.
- Due to differences in mechanism of algorithms, we train each algorithm for one day instead of some fixed number of iterations.
- Original settings of Hausknecht and Stone (for DDPG) [2] and Duan et al. [7] (For NPOs) were used. We then altered the neural network structure and hyper parameters with the following rationalities:
  - We halfed the size of all layers in the neural networks of the original DDPG unit. The original amount of nodes are very large compared to other works of similar tasks in MuJoCo.
  - We have tried to increase the number of neurons in the NPO based algorithms since this task is much more complex than the previous tasks. Also our initial experiments using settings based on Duan et al.'s MuJoCo settings does not provide very convincing results.
  - We tried to remove a non-intuitional part from Hausknecht and Stone's work. In order to bound the output of action parameters, they have used a reversing gradient technique to change the direction of weight update under certain conditions. According to their experiment results, this technique is an important and essential part that ultimately makes their method worked. However, to intentionally tweak with gradients during weight update is not normally seen in training of neural networks. Therefore, we test if we can use some generalized methods to bound the parameters
- Every agent was trained three times and the final average reward for the best agent was listed in Table 4.

5. Results

Results of the experiments are shown in Table 4. We carried out five kinds of different experiments with each agent being trained for three times and we use the best result out of the three. The first agent DDPG1 uses DDPG with two dense networks (actor and critic), as same as Hausknecht and Stone. The agent did not perform as well as the previous work stated and it was outperformed by an agent using the same settings but half-sized neural networks. The best performer is DDPG2 using a relatively simpler network. This agent was able to score a goal during the one day training and got the best average reward.

However, drawbacks of DDPG were also observed in the experiments. Similar behavior have been described by Duan et al. in their experiment results section for DDPG. Learning of DDPG does not always advance towards positive reward stably. When the algorithm searches for optimum policy it always gets stuck in local optimum due to inaccurate update step length. Also, when an bad policy is learned

by the algorithm, game trajectories of bad moves would be collected. Updating the agent further using these trajectories would likely to generate a worse policy. Eventually the agent has a chance to break down.

Special attention needs to be paid on the **DDPG_NoRevGrad** case. In this test case we removed the reversing gradient part of the DDPG implementation proposed by [2]. In order to bound the parameters we used a sigmoid activation layer to output the power parameters (ranged between 0 to 100). The output is scaled up to fit the boundary of the parameters. We then used the same method on a tanh activation layer to output the degree parameters (ranged from -180 to 180). However this modification of the previous algorithm does not seem to success. In the start of the training process the agent picks a route towards positive rewards. After around 200 iterations of training the agent starts to produce extreme parameter values close to the boundary of the parameters. We can see that with naive bounding of the parameters the neural network does not produce suitable parameter values to gain high reward. We therefore proved that although unnatural, the bounding of parameter outputs are required. The current model of neural network seems not capable of dealing with parameter boundaries by its own.

Compared to DDPG, NPO-based algorithms did not perform well under the existing settings. Signs of learning have been found as the agents are able to achieve small increase in the average reward. After one day of training, TNPG1 and TRPO1 achieved similar performance in terms of average reward and max reward.

Inspired by [7], we also used perplexity, a standard that is often used in natural language processing models to evaluate how well the NPO agents understand the environments. Perplexity describes the branching factor when a agent makes decisions on the next action. Therefore, the smaller the perplexity value, the better the agent understands how to deal with the current situation in a game. Notable jumps of perplexity is observed during initial training training. The agents suffer drop of average reward when these perplexity jumps occurs. Since perplexity reflects how well does the agent consider itself understand the environment. These sudden surges of perplexity shows that the agents continuously underestimates the complexity of the environment.

When further training is applied, perplexity normally gradually goes down again with the observable re-rise on average reward of the agent. This surge and drop can be observed multiple times during training. In one way, this behavior shows that more time is required to train these NPO-based agents so that the perplexity goes stable and agents are understanding the environment well enough. The other way of thinking about perplexity rise is that we can alter the batch size for each batch update for the NPO algorithms so these algorithms understand more about the environment in each update run.

We stabilized the perplexity during training to make it does not fluctuate too much by setting the batch size for each iteration of the algorithm larger. This is shown in experiment case TRPO3. However the final training outcome does not change too much. We need to do some more experiments before deciding if the convergence is finished for NPO based algorithms.

We have also found that NPO based algorithms with deeper and wider neural network does not converge at all. When training with large neural networks（experiment case TRPO2）, perplexity of policy get stuck in the beginning of the experiment and no obvious sign of positive reward increase. The ability of these algorithms to train larger networks for complex problems is still in doubt.

6. Conclusions and Future Work

In this work we investigated an interesting environment, Half Field Offense (HFO) for the latest deep reinforcement learning algorithms. We use HFO as a benchmark environment as it has multiple characteristics that may test the limits of these algorithms. These includes large state and action space, parameterized action space and potentially multi-agent testing capabilities.

The parameterized action space in the HFO environment is what makes the environment special and stands out. We can find this property in many real-life tasks that can be potentially completed by deep reinforcement learning agents. During the experiments we have observed that learning to produce correct parameters is a very difficult task for most of the existing models.

We were able to observe positive updates towards higher reward for all the tested algorithms rational hyper parameter and neural network settings. Although DDPG sometimes displays unstable fluctuations in terms of average reward and got stuck into local optimum or even break down at times, one agent powered by DDPG still got the best performance out of the five. NPO based algorithms showed great potential of learning a good policy if they converge. However significant increase of training time is required to obtain better results.

We have also observed that DDPG is unable to learn to produce rational parameters if no special procedure is done to prevent parameters going to extreme values during gradient descent. In [2] a extra step of reversing gradient is proposed and proved to be working. However, this method is not very intuitive and does not utilize the power of deep learning throughly. Also checking every gradient update during training to see if reverse is required is computationally expensive. Therefore in the future we need to figure out a vectorized method to update the parameters within boundaries.

Unfortunately, due to the long time requirement for training each agent and the limited computation resource. We are not able to tune and test each algorithm to the point that they converge and obtain stable scoring behavior. Also we do not have time to test some other existing algorithms that was mentioned by Duan et al. In the future we aim to

Algorithm & No.	Neural Network Structure	Training Time	Perplexity	Able to Score	Max Reward	Average Reward
DDPG1	Dense 1024x512x256x128	24 hours	N/A	False	1.55	0.82
DDPG2	Dense 512x256x128x64	24 hours	N/A	**True**	**8.03**	**2.53**
DDPG_NoRevGrad	Dense 512x256x128x64	24 hours	N/A	False	0.93	-0.02
TRPO1	Dense 100x50x25	24 hours	Starting: ≈80000 Maximum: ≈300000 Minimum: ≈60000	False	1.35	0.08
TRPO2	Dense 1024x512x256x128	24 hours	Starting: ≈80000 Maximum: ≈80000 Minimum: ≈80000	False	0.13	0.00
TRPO3	Dense 100x50x25	24 hours	Starting: ≈80000 Maximum: ≈80000 Minimum: ≈30000	False	1.28	0.10
TNPG1	Dense 100x50x25	24 hours	Starting: ≈80000 Maximum: ≈150000 Minimum: ≈30000	False	1.32	0.14

Table 4: Results for running different algorithms on the HFO environment.

work on the following milestones to make deep reinforcement learning development in HFO contributes more to the how society of deep reinforcement learning and take one more step further into real-life applications of deep reinforcement learning in complex tasks:

- Further training of existing algorithms with existing hyper parameters to get better performance and observe convergence.
- Tuning of existing algorithms to get better performance.
- Experiment with more neural network structures for NPO based algorithms to see how they can handle large and deep neural networks. Also, we want to see the increase of size and depth of the neural networks affect the performance of these algorithms.
- Apply DDPG or other algorithms without the gradient reversing technique.
- Apply more available algorithms in the HFO task.
- Test existing algorithms on other more complex HFO tasks.
- Design an algorithm to play the game with multiple-agent scenarios. Beyond the above goals, we want to encourage game AI and deep reinforcement learning researchers to look further than existing experimental environments like Atari or MuJoCo. At the moment many exciting and challenging video game environments are being announced and released almost every year. Many of these games are like

References

[1] Matthew Hausknecht, Prannoy Mupparaju, Sandeep Subramanian, Shivaram Kalyanakrishnan, and Peter Stone *Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork*. In AAMAS Adaptive Learning Agents (ALA) Workshop, Singapore, May 2016.

[2] Matthew Hausknecht, Peter Stone *Deep Reinforcement Learning in Parameterized Action Space*. In ICLR 2016, Feb 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, Demis Hassabis. *Human-level control through deep reinforcement learning*. Nature 518, 529–533, Feb. 2015.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra *Continuous control with deep reinforcement learning*. In ICLR 2016 , Feb 2016.

[5] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel *Trust Region Policy Optimization*. In ICML 2015, Feb 2015.

[6] Sham Kakade *A Natural Policy Gradient*. In NIPS 2002

[7] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, Pieter Abbeel *Benchmarking Deep Reinforcement Learning for Continuous Control*. In ICML 2016, May 2016

[8] Matthew Hausknecht *RoboCup 2D Half Field Offense Technical Manual*

[9] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa *Simultaneously Learning and Advising in Multiagent Reinforcement Learning*. In AAMAS 2017 Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, May 2017

[10] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. *Making friends on the fly: Cooperating with new teammates*. Artificial Intelligence, October 2016.

[11] Emanuel Todorov, Tom Erez and Yuval Tassa *MuJoCo: A physics engine for model-based control*. In IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[12] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba *OpenAI Gym*. arXiv preprint, arXiv:1606.01540, 2016.

[13] Szita, I. and Lorincz, A. *Learning Tetris using the noisy cross-entropy method*. Neural Comput., 18(12):2936–2941, 2006.

[14] Hansen, N. and Ostermeier, A. *Completely derandomized self adaptation in evolution strategies.*. Evol. Comput., 9(2):159–195, 2001.

[15] Kulkarni, Tejas D and Narasimhan, Karthik and Saeedi, Ardavan and Tenenbaum, Josh *Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation* In NIPS 2016, 2016.

[16] Russell Kaplan, Christopher Sauer, and Alexander Sosa. *Beating atari with natural language guided reinforcement learning*. In arXiv:1704.05539, 2017.

[17] Riashat Islam, Peter Henderson, Maziar Gomrokchi and Doina Precup. *Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control* In arXiv: 1708,04133.pdf, Accepted to Reproducibility in Machine Learning Workshop, ICML 2017