Collaboration and Competition

Christoffer Edlund

Abstract—In this project we train two unity agents to play a collaborative game of tennis with each other. The goal of the two agents is to pass the ball back and forth over the net between one another, if the ball hits the ground or goes out of bound the agents get punished by a reward of -0.01 while each pass of the ball over the net gives the agents a reward of +0.1. The agents managed to solve the environment in 4297 episodes using the MADDPG algorithm, meaning that they got an average score of +0.5 over 100 consecutive episodes.

Index Terms—Reinforcement Learning, Udacity, Deep Learning, DDPG, MADDPG, Actor-critic.

1 Introduction

The Collaboration and Competition project involves training two deep reinforcement learning agents to control tennis rackets with the task of bouncing a ball back and forth over a net between each other for as long as possible. The agents receives a positive reward each time they get the ball over the net and get punished if either of the agents drop the ball or if it goes out of bounds, see Figure 1.

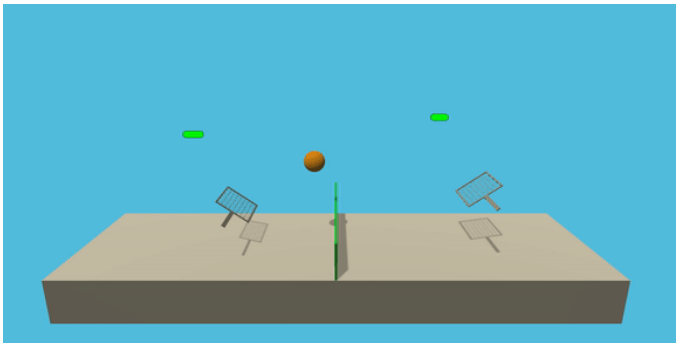

Fig. 1. Example view of the multi-agent environment utilizing the Unity Engine. Two agents controlling a tennis racket each are tasked with passing the ball between one-another over the tennis net.

To train the agents the Multi-agent Deep Deterministic Policy Gradient (MADDPG) algorithmen was used [1]. MADDPG extends the Deep Deterministic Policy Gradient (DDPG) actor-critic method [2]. The original DDPG utilizes two neuronal networks, one that estimates the future rewards of the agents actions based on current state, and another that is tasked to learn the optimal policy (best action based on current state). Multi-agent DDPG extends DDPG by training decentralized agents to achieve the optimal policy in a multi-agent environment while guided by centralized critics that is aware of the whole environment as well as all agents in it. This method allows for training of collaborative agents as in this project as well as competitive and mixed environments.

2 Actor-critic algorithms

In value-based methods the agent is trying to learn how to predict the Q-values, (the future reward), associated with the possible actions given current state. The agent will use a pre-defined policy of how to select its next action based on the given Q-values, an example being the epsilon-greedy policy or just taking the action which will yield the best predicted reward. This can be contrasted with policy-based methods were the agent is tasked to learn the optimal policy, meaning that the agent will try to predict the best action to take given current state directly.

Actor-critic algorithms is a combination of policy- and value-based models, were the actor is tasked with finding the optimal policy and the critic will evaluate and estimate the future rewards. The upside of this joint combination is that we can train an agent to find the optimal policy in a temporal difference (TD) learning setting. In some policy-based models we need to gather up a series of experience, or a so called trajectory, so that we can calculate the discounted future reward associated with the agent selecting a certain policy. By using deep Q-learning (our value-based method) we can instead let the critic learn to approximate the future rewards and use that to train the actor with any state-action pair. All we need to do is let the critic give its best guess as to what kind of reward the state-action pair will yield.

3 Deep Deterministic Policy Gradient

The solution of this environment is based on the deep deterministic policy gradient (DDPG) method [2]. This reinforcement learning algorithm is an actor-critic method were the actor takes in a state and predicts next action, and the critic network takes in a state-action pair and predict the expected Q-value.

The critic learns in a similar fashion as other deep q-learning algorithms [3] except that it only predicts one Q-value while some other value-methods predicts all possible action-value pairs. The actor learns by making an action prediction based on the current state and feeding the state-action pair to the critic, that will provide an estimated future-reward for that action. By using gradient decent on the negative reward (in effect doing gradient ascent) the actor will corrects its action prediction based on the critics ”best guess” of how the actor can achieve a better reward

next time a similar state is observed.

DDPG utilizes target networks like the double DQN algorithm, meaning that there exists a copy of the actor and critic networks that is used to stabilize the training [4]. The target networks are used to generate the Q-targets when updating the critic network. The actor target network predicts next action that is used as input together with next states to the target critic that will generate the Q-target.

In the original DDPG algorithm they also a prioritized experience replay (PER) buffer while learning [5], this is something we skip in our implementation of the algorithm. In a normal replay buffer there is a equal probability that the agent will pick any collected experience to learn from,

4 Multi-agent DDPG

In the MADDPG setting the DDPG algorithm is extended to accommodate training of multiple independent agents acting in a collaborative, competitive or mixed environment [1]. This is done by extending the critic networks so that they are able to observe the whole environment as well as the action of other agents, while the actors still only have access to their own perceived state of the environment. Since the critics task is to train the actor by estimating the future rewards for a given state-action pair, and wont be present during the inference phase, it makes sense to give the critic as much information as possible to improve its estimate of the Q-value during training. Creating decentralized training of the agents and centralised training of the critics, to support the agents acting in a decentralized manner in a given environment.

5 Environment

The tennis environment consists of two tennis rackets located on each side of a net and a ball that can be manipulated by the rackets. The agents are in control of the rackets and control them via a vector $\mathbb{R}^2 \in [-1, 1]$ corresponding to moving towards or away from the net as well as jumping. The environment state is observed via a vector $\mathbb{R}^8$ that corresponds to the position and velocity of the ball and racket. The aim of the agents is to collaborate with one another and pass the ball back and forth without it touching the ground.

To model this joint goal, the agents receives a reward of $+0.1$ for each time the ball goes over the net and -0.01 when the ball hits the ground or goes out of bound. We consider the environment solved once the agents have learned to cooperate and can together generate an average score of $+0.5$ over 100 consecutive episodes, taking the maximum score per episode from the two agents.

6 Solution

The network architectures was based on the paper introducing the MADDPG algorithm [1], both the actor and critic network was constructed with 2 hidden layers with 400 and 300 neurons respectively. For the critic network we concatenate the actions from the actor network at the second hidden layer.

The learning rates for both the actor and critic was set to $1e^{-4}$ and $1e^{-3}$ respectively, the training was done with a batch size of 200, discount factor γ of 0.995, and the size of the replay buffer was initialized to $1e^5$. For the training we found the best scheme of training is to update the weights 3 times each 4th time step of interacting with the environment. The target networks was updated utilizing soft-updates with a rate of $\tau = 1e^{-3}$. To randomize the actions for exploitative purposes Ornestein-Uhlenbeck noise was used with $\sigma = 0.2$, $\theta = 0.15$ and $\mu = 0.0$.

7 Results

Our maddpg algorithm managed to solve the tennis environment in 4297 episodes as can be seen in Figure 2

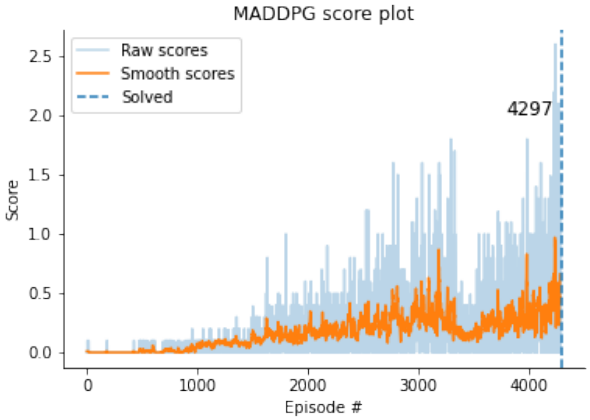

Fig. 2. Training plots for a multiple agents playing tennis using the MADDPG algorithm to solve the simulated environment in 4297 episodes. The environment is considered solved when the agent has reached an average score of $+0.5$ over 100 episodes. The plot shows booth a smoothed and the raw score plots.

8 Conclusion / Future work

We can conclude that the MADDPG algorithm worked for solving the environment in 4297 episodes and can work as a good baseline to evaluate further methods and adjustments.

This performance of these models would probably improve by an extensive hyperparameter search as well as the usage of a prioritized experience replay buffer [5], a technique that was part of the original MADDPG algorithm but not in this implementation.

References

[1] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," p. 12.

[2] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "DISTRIBUTED DISTRIBUTIONAL DETERMINISTIC POLICY GRADIENTS," p. 16, 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[4] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016. Number: 1.

[5] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv:1511.05952 [cs]*, Feb. 2016. arXiv: 1511.05952.