Continuous Control

Christoffer Edlund

Abstract—In this project we train a unity agent tasked with controlling a double jointed arm to reach out and follow a moving sphere. The training was done using reinforcement learning and the deep deterministic policy gradient actor-critic algorithm. We solved the environment in 861 episodes for a single agent and in 120 episodes when using a distributed environment with 20 agents.

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

1 Introduction

The Continuous Control project involves training a reinforcement learning agent to control a double-joint arm with the task of moving it into a target location. In this case the target location is a moving sphere that circles the agent, and the agent receives a reward each frame it manage to keep the tip of its arm inside of the circle, see Figure 1.

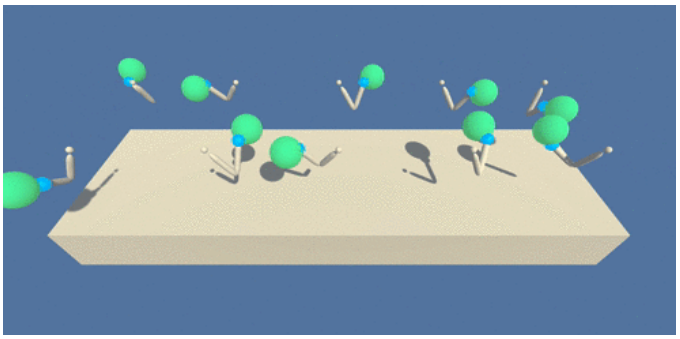

Fig. 1. Example view of the multi-agent environment utilizing the Unity Engine. Multiple robotic arms that recives a reward each frame they are inside the moving green spheres.

The agents learning algorithm is based on actor-critic method called Deep Deterministic Policy Gradient (DDPG) [1]. 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).

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 [1]. 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 [2] 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 [3]. 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 [4], 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 Environment

The environment consists of double-joint arms with "hands" on the top. The agent are in control of the arms and control them via a vector $\mathbb{R}^4 \in [-1, 1]$ corresponding to the torque applied to the two joints. The environment state is observed via a vector $\mathbb{R}^{33}$ that corresponds to the position, rotation, velocity and angular velocities of the arm. There is a sphere moving around in the environment, see Figure.1, and the aim of the agent is to move the arm in such a way that the hand is inside the sphere. To model this goal, the agent receives a reward of $+0.1$ for each time step the hand is inside the target sphere.

In this project we explore two different settings of the environment, one with a single agent and one with 20 identical agents learning in a distributed fashion. For these experiments we defined an episode as 1000 time steps, and let the agent train for a maximum of 1000 episodes to solve the environment. The environment was considered solved once the agent manage to collect a average score of +30 over 100 consecutive episodes. For the multi-agent environment it was instead the average score of the all the agents that had to have an average over or equal to 30 over the 100 episodes.

5 Solution

The network architectures was based on the paper introducing the DDPG 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 $1e^{-3}$, and the training was done with a batch size of 128, discount factor γ of 0.99, 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 10 times each 20th time step of interacting with the environment. The target networks was updated utilizing soft-updates with a rate of $\tau = 1e^{-3}$.

6 Results

The single agent environment was solved in 861 episodes as can be seen in Figure.2.

The multi-agent environment was solved after each agent experienced 120 episodes, meaning that we have gathered a total of 2400 experiences from the 20 agents, the plot can be seen in Figure.3.

7 Conclusion / Future work

We can conclude that the DDPG algorithm worked for solving the environment and can work as a good baseline to evaluate further methods and adjustments. It looked like the multi-agent environment was able to learn in fewer episodes (120) compared to the single agent (861) by leveraging the experience of all 20 agents at once. But it is worth noting that when combining the experiences of all agents, we end up with 2400 recorded episodes which is higher than the amount of episodes the single agent required.

This performance of these models would probably improve by an extensive hyperparameter search as well as the usage of a experience replay buffer as per the original DDPG algorithm.

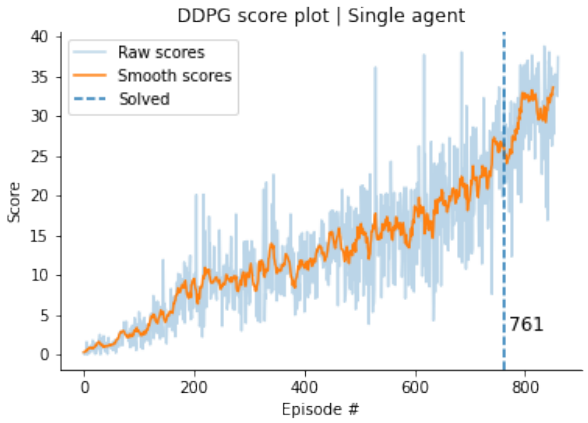

Fig. 2. Training plots for a single agent using the DDPG algorithm to solve the simulated environment in 861 episodes. The environment is considered solved when the agent has reached an average score of 30+ over 100 episodes. The plot shows booth a smoothed and the raw score plots.

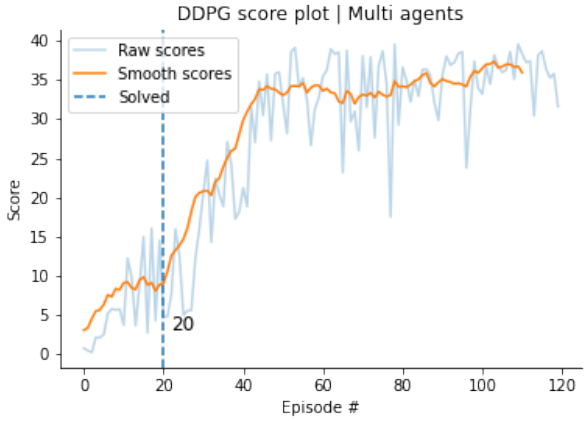

Fig. 3. Training plots for a distributed training of 20 agents using the DDPG algorithm to solve the simulated environment in 120 episodes. The environment is considered solved when the agents the average score from each agent was an average value of 30+ over 100 episodes. The plot shows booth a smoothed and the raw score plots.

References

[1] 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.

[2] 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.

[3] 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.
[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv:1511.05952 [cs]*, Feb. 2016. arXiv: 1511.05952.