

Notes for the Collaboration and competition project

Raphael Gross

2018
November

1 Introduction

This is the third project of the Udacity DRL course. My goal for this project is to get a better understanding of multi-agents environments using deep reinforcement learning by implementing some of last variation of Deep Deterministic Policy Gradient (DDPG) in pytorch. In order to achieve my goal, I describe briefly the main features of the DDPG methods, the Tennis environment used for testing the algorithm and finally, I show the results I obtained.

- ddpg: one agent, playing for both side. - MA-DDPG: two agents with no shared memory. - MA-DDPG: two agents with shared memory for the critic only. - MA-DDPG: two agents with shared memory for critic and actor

2 State of the Art for DDPG

I used the DDPG algorithm to solve the Tennis environment. DDPG is a model-free, off-policy actor-critic algorithm that enables solving continuous action environment. DDPG is based on the deterministic policy gradient or DPG [6]. DDPG has a parametrized actor function $\mu(s|\theta^\mu)$ which specifies the current policy by deterministically mapping states to a specific action while the critic $Q(s, a)$ is learned using the Bellman equation.

DDPG uses some of the advantages of DQN such as sample replay and the target network. Sample replay reduces correlations between sample. The

replay buffer is constituted of tuples (s, a, r, s') with s the observations, a actions, r rewards and s' the new states. I stock the interaction between agents and the environment in the same replay buffer without distinction.

The DDPG model like DQN [5] uses a target network adapted to actor-critic and soft target updates for the network weights. We have an actor target μ' and a critic target Q' . These two points increase the stability of the network and let it learn the action-value function.

The principal issue in continuous action space is exploration. I define the exploration policy as the sum of the policy μ and some noise function \mathcal{N} obtained using the Ornstein-Uhlenbeck process.

3 Testing environment

The environment I used to train and test my agent is called Tennis. In this environment, two agents play tennis while controlling rackets. The objective is to make as many passes as possible. To achieve this objective, they need to pass the ball over a net without letting the ball touch the ground or shooting the ball out of the boundary. It is therefore a collaborative environment. For each pass done successfully, the agent get a reward of +0.1. If the agent fault the ball a reward of -0.01 is attributed.

The observation space is composed of 24 variables and each action is determined by a vector of dimension 2 corresponding to going forward or backward and jump. The action vector values are between -1.0 and 1.0.

For this project, I will use the multi-agent Unity environment. This environment is composed of 2 identical agents. Each of them is on one side of a tennis court. The task is episodic. After each episode, each agent gets a reward without discounting. In order to solve the environment, the average maximum score needs to be greater than 0.5 over 100 consecutive episodes.

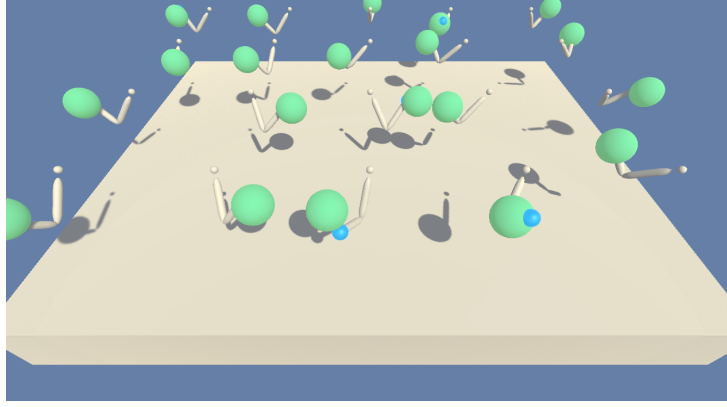


Figure 1: Reacher environment

4 Results

4.1 DDPG model: naive approach

The DDPG method is composed of two networks one for the actor and one for the critic. I call this version the naive approach as one agent is playing with himself.

4.1.1 Actor

The actor network is composed of linear layers:

- $fc1 : nn.Linear(state_size, fc1_units)$
- $fc2 : nn.Linear(fc1_units, fc2_units)$
- $fc3 : nn.Linear(fc2_units, action_size)$

The weight for the first two hidden layers is initialized using a Xavier initialization. The number of input and output nodes for the hidden layers is defined by the next parameters:

- $state_size = 24$
- $fc1_units = 256$
- $fc2_units = 128$

- $action_size = 2$

In the forward pass, $fc1$ and $fc2$ are combined with a *ReLU* rectifier. Finally, the policy is given after injecting $fc3$ into a tanh logistic sigmoid function.

4.1.2 Critic

The critic-network is composed of linear layers:

- $fc1 = nn.Linear(state_size, fc1_units)$
- $fc2 = nn.Linear(fc1_units + action_size, fc2_units)$
- $fc3 = nn.Linear(fc2_units, 1)$

The number of input and output nodes for the hidden layers is defined by the next parameters:

- $state_size = 24$
- $fc1_units = 256$
- $fc2_units = 128$
- $action_size = 2$

The weights for the first two hidden layers are initialized using a Xavier initialization. The architecture of the critic is a bit unusual. The critic is used to estimate $Q(s, a(\mu))$. But, the action contribution is added to the network only after the first hidden layer.

The authors of the DDPG method [6] decided to concatenate the output of the first layer with the action just before the second hidden layer. One possible reason for this choice is to reduce the impact of the critic gradient on the actor computation during the backpropagation.

- $x_s = F.relu(self.fc1(state))$
- $x_s_a = torch.cat((x_s, action), dim = 1)$
- $x_s_a = F.relu(self.fc2(x_s_a))$
- $Q_s_a = self.fc3(x_s_a)$

4.2 Parameters

The best result I obtained for the DDPG method was with the next parameters values:

- MEMORY SIZE = $\text{int}(1e5)$ (replay buffer size)
- BATCH SIZE = 128 (minibatch size)
- GAMMA = 0.99 (discount factor)
- TAU = $1e-3$ (parameter value used for the soft update of the target weights)
- LR_ACTOR = $8e-5$ (learning rate for actor)
- LR_CRITIC = $8e-5$ (learning rate for critic)
- UPDATE EVERY = 1 (how often the target network is updated)

4.3 Score

The maximum score is defined as the highest score between agents. The score is the sum of all the rewards an agent obtained during an episode. For the naive approach, we have technically one agent playing with himself. So we still have two scores: one for "left side" and one for the "right side" of the net.

In Figure 2, I plotted the maximum score obtained by the agents at each episode. To solve the environment, the agent need to have an average score of 0.5 or above over 100 episodes. The algorythm took 1972 episodes to solve the environement with an average score for the last 100 episodes of: 0.5029000074975193. The score for the last episode is 1.00 which corresponds to 10 sucessful exchange.

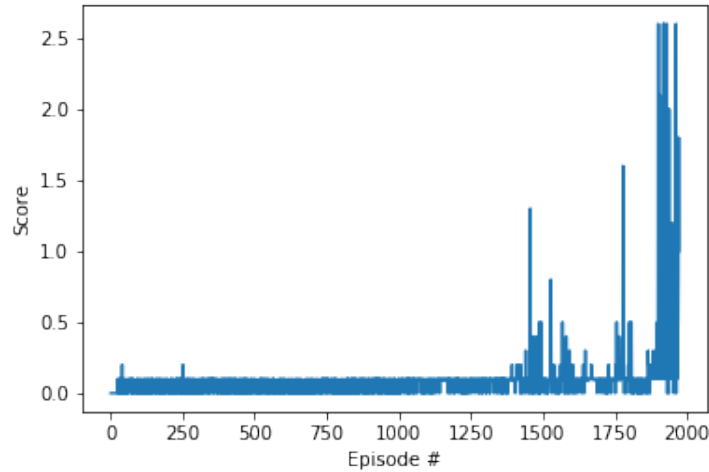


Figure 2: Maximum score par episode

4.4 Future improvement

For now, I only covered the DDPG [3] algorithm. But, my ambition is to finish the PPO method [2] I started to implement for continuous action and also look into A3C [4], A2C [4] and D4PG [1]. i will also try to solve other environment such as the Crawler environment.

References

- [1] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy P. Lillicrap. Distributed distributional deterministic policy gradients. *CoRR*, abs/1804.08617, 2018.
- [2] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *CoRL*, volume 87 of *Proceedings of Machine Learning Research*, pages 617–629. PMLR, 2018.
- [3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

- [4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.
- [5] 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, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015.
- [6] David Silver. Deterministic policy gradient algorithms, 2014.