Notes for the continous control project

Raphael Gross

2018
November

1 Introduction

This is the second project of the Udacity DRL course. My goal for this project is to get a better understanding of continuous control with deep reinforcement learning by implementing the Deep Deterministic Policy Gradient (DDPG) algorithm in pytorch. In order to achieve my goal, I describe briefly the main features of the DDPG method, the Reacher environment used for testing the algorithm and finally, I show the results I obtained.

2 State of the Art

I used the DDPG algorithm to solve the Reacher 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 constitued 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 mu' 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 Reacher. In this environment, the agent control a double-jointed arm. The arm needs to reach a moving target location. For each step the agent manages to touch the target location, a reward of +0.1 is provided. Thus, the goal of my agent is to maintain as long and often as possible the arm hand on the target location.

The observation space is composed of 33 variables corresponding to the position, rotation, velocity, and angular velocities of the arm. Each action is determined by a vector of dimension 4 corresponding to the torque applied to the two joints. 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 20 identical agents and each of them has its own copy of the environment. The task is episodic. After each episode, each agent gets a reward without discounting. In order to solve the environment, the average score (all agents) needs to be greater than 30 over 100 consecutive episodes.

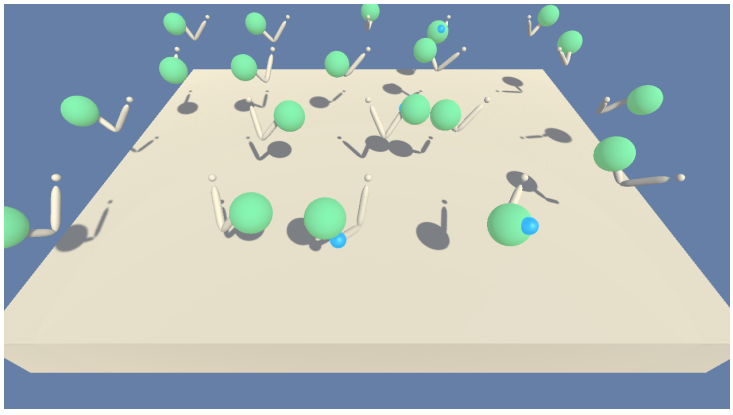

Figure 1: Reacher environment

4 Results

4.1 Models

he DDPG method is composed of two networks one for the actor and one for the critic.

4.1.1 Actor

The actor-network is composed of linear layers:

4.1.2 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:

- $state\_size = 33$
- $fc1\_units = 256$
- $fc2\_units = 128$
- $action\_size = 4$

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.

4.1.3 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:

- $state\_size = 33$
- $fc1\_units = 256$
- $fc2\_units = 128$
- $action\_size = 4$

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 = 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

In Figure 2, I plot the score fo obtained by each agent at each episode. The score corresponds to the sum of all the reward an agent obtained during an episode. The algorithm took 135 episodes to converge.

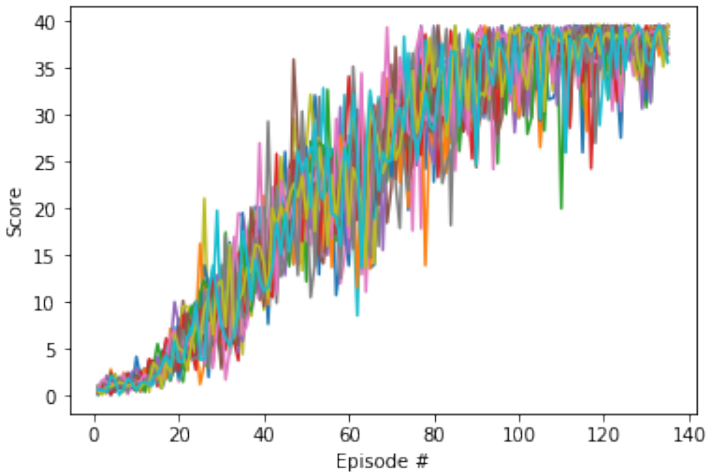

Figure 2: Score for each agent

In Figure 3, I display the mean score at each episode.

4.4 Score

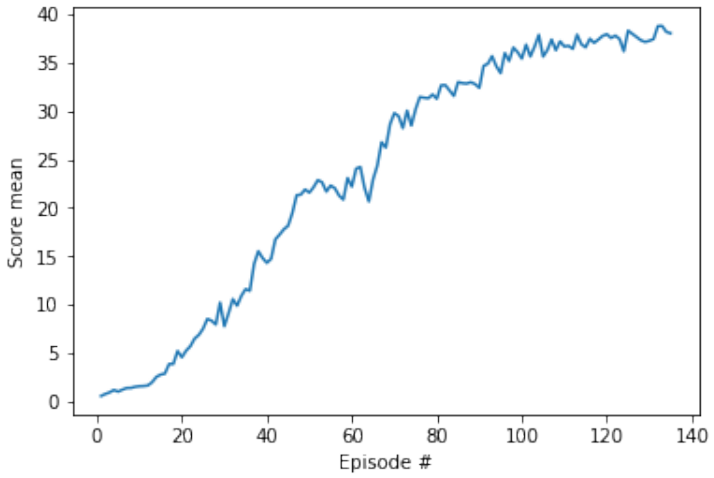

Figure 3: Average score

4.5 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, Dharshan 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.