Project 2. Continuous Control

Zhamila Issimova

January 3, 2019

Abstract

In this report implementation of DDPG algorithm for continuous control problem is described. It contains brief information about this deep reinforcement learning technique as well as design choices which were made to train machine learning agent in Unity environment.

1 Introduction

The goal of the project is to maintain double-jointed arm position at the target location for as many time steps as possible. The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Hence, this is a continuous reinforcement learning task, which cannot be solved by famous DQN algorithm [1], since discretization of continuous space is computationally costly. Therefore, Deep Deterministic Policy Gradients (DDPG) algorithm [2] is used to solve this Reacher [3] environment problem. This DDPG algorithm has actor-critic architecture, where actor learns policy function and controls how our agent acts. Critic, on the other hand, learns value function and measures how good these actions are. Therefore, actor-critic algorithms combine value and policy gradient approaches.

2 Methodology

2.1 Environment

Continuous control problem is represented in Reacher environment, which is provided by The Unity Machine Learning Agents Toolkit (ML-Agents). Since the goal is to maximize the score and at the same time maintain robotic arm position at the target location, a reward of +0.1 is provided for each step that the agent's hand is in the goal location. State space has 33 dimensions which includes agent's position, rotation, velocity, and angular velocities of the arm. . Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.
In this project second version of the Unity environment was used, which contains 20 identical agents, each with its own copy of the environment. Since algorithm used multiple (non-interacting, parallel) copies of the same agent to distribute the task of gathering experience, the learning process speed was increased.

```
Episode 170, Average Score: 4.73, Max: 10.34, Min: 2.51
Episode 180, Average Score: 5.01, Max: 22.90, Min: 3.05
Episode 190, Average Score: 5.31, Max: 10.31, Min: 2.35
Episode 200, Average Score: 5.62, Max: 14.35, Min: 0.76
Episode 210, Average Score: 5.96, Max: 13.37, Min: 3.31
Episode 220, Average Score: 6.28, Max: 9.59, Min: 0.00
Episode 230, Average Score: 6.64, Max: 13.37, Min: 0.09
Episode 240, Average Score: 6.94, Max: 17.16, Min: 1.04
Episode 250, Average Score: 7.26, Max: 14.41, Min: 5.10
Episode 260, Average Score: 7.65, Max: 12.01, Min: 4.04
Episode 270, Average Score: 7.93, Max: 14.40, Min: 5.97
Episode 280, Average Score: 8.11, Max: 27.39, Min: 4.11
Episode 290, Average Score: 8.21, Max: 11.83, Min: 4.63
Episode 300, Average Score: 8.38, Max: 14.75, Min: 5.93
Episode 310, Average Score: 8.48, Max: 15.32, Min: 0.00
Episode 320, Average Score: 8.67, Max: 16.75, Min: 6.03
Episode 330, Average Score: 8.83, Max: 16.37, Min: 5.77
Episode 340, Average Score: 9.13, Max: 15.26, Min: 6.48
Episode 350, Average Score: 9.34, Max: 30.14, Min: 6.93
Episode 360, Average Score: 9.52, Max: 15.67, Min: 5.49
Episode 370, Average Score: 9.64, Max: 14.70, Min: 5.24
Episode 380, Average Score: 9.84, Max: 17.77, Min: 7.59
Episode 390, Average Score: 10.09, Max: 15.19, Min: 3.75
Episode 400, Average Score: 10.25, Max: 15.60, Min: 5.37
Episode 410, Average Score: 10.32, Max: 13.31, Min: 6.47
Episode 420, Average Score: 10.39, Max: 33.47, Min: 7.33
Episode 430, Average Score: 10.48, Max: 23.04, Min: 6.32
Episode 440, Average Score: 10.46, Max: 14.80, Min: 6.12
Episode 450, Average Score: 10.46, Max: 14.91, Min: 6.56
Episode 460, Average Score: 10.42, Max: 15.06, Min: 6.88
Episode 470, Average Score: 10.51, Max: 28.01, Min: 1.87
Episode 480, Average Score: 10.47, Max: 13.36, Min: 5.38
Episode 490, Average Score: 10.36, Max: 23.54, Min: 4.90
```

Figure 1: Slow learning with actor lr=0.0001

2.2 DDPG algorithm

Since action space is continuous and the goal is to maximize output, DDPG perfectly fits to solve this problem. DDPG algorithm was taught earlier on the program and its solution implementation files ($ddpg_agent.py$ and $model.py$) were taken with some changes from "ddpg-pendulum" folder from Udacity Deep Reinforcement Learning folder on Github. In this project version of DDPG algorithm actor-critic architecture with separate neural networks for actor and critic was implemented just the way it was described in DDPG paper. In particular, experience replay buffer, local and target networks, soft update of these networks and batch normalization in layers were realized. Ornstein-Uhlenbeck process was also added as exploration noise and weights were initialized from uniform distributions in order to ensure the initial outputs of the policy and value estimates were near to zero. Additionally, in initialization step, hard update of local and target networks was done in order to ensure that they were identical at the beginning. However, L2 regularizer of weight decay described in the paper was not included into this project implementation.
Since 20 agents were used for distributed learning and experience augmentation, "step" function in class "Agent" was modified. In order to further speed up learning process, exploration noise influence was reduced as more and more actions were taken. Furthermore, for the same reason learning rate was made the same for actor and critic networks (lr=0.001), which significantly increased learning process without harming performance and stability. As it can be seen from Figure 1, with actor network learning rate of 0.0001, overall algorithm speed is really slow.In terms of hyperparameters, batch size was increased to 256, and the rest was the same as in DDPG paper.

2.3 Deep Neural Network

Since it is an actor-critic algorithm, separate neural networks were created for actor and critic. Actor network consists of 2 hidden fully connected layers

```
Episode 148, Average Score: 28.87, Max: 35.49, Min: 24.02
Episode 149, Average Score: 29.12, Max: 38.35, Min: 29.67
Episode 150, Average Score: 29.36, Max: 39.16, Min: 28.05
Episode 151, Average Score: 29.60, Max: 38.33, Min: 34.04
Episode 152, Average Score: 29.85, Max: 38.25, Min: 34.82
Episode 153, Average Score: 30.05, Max: 39.02, Min: 33.42
Episode 153     Average Score: 30.05
```

Figure 2: Environment solved in 153 episodes

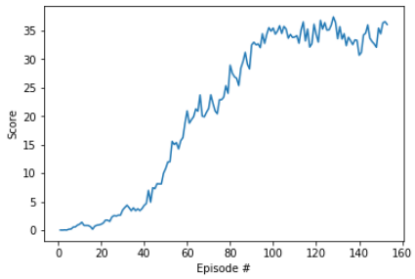

Figure 3: Learning curve

with 400 and 300 units respectively. Each layer contained batch normalization. Hidden layers outputs were passed through ReLU function and final output was passed through tanh function to obtain action vector. As it was stated above, initial weights were set close to zero.
Critic network also consists of 2 hidden fully connected layers with 400 and 300 units respectively, but on the second hidden layer actions are added to the input, since critic needs to evaluate how good actor network is performing state-action mapping. Batch normalization is used only before first layer, since later actions and states are mixed together and data is no longer homogeneous. Just like in actor network, in critic network we have weights initialization in the beginning and hidden layers outputs were passed through ReLU function.

3 Results

The number of episodes were set as 1000, and length of episode was set as 3000 steps. The environment was solved in 153 episodes as it can be seen from Figure 2. The problem was considered as solved if average score over 100 episodes is higher than 30. The average score during whole training can be seen on Figure 3.

4 Conclusion

In this project extensive experience with DDPG algorithm was obtained. DDPG was adapted to Unity "Reacher" environment. The goal of continuous control project was solved and agents were trained to have average score more than 30. In future, it is possible to extend this work and make agent to learn directly from visualized input using only raw pixels. Also weight decay mentioned in the original paper could be implemented.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. 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, no. 7540, pp. 529-533, 2015.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, "Continuous control with deep reinforcement learning", *CoRR*, 2015.

[3] "Unity-Technologies/ml-agents", GitHub, 2019. [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Examples.mdreacher. [Accessed: 03- Jan- 2019].