Udacity DRLND Project #2 Continuous Control

Jing Zhao 10/29/2018

Introduction

In this report, I described in detail the implementation of a deep deterministic policy gradient (DDPG) algorithm for solving the continuous control (Reacher) problem in the Unity ML-Agents environment. I first introduced the environment followed by a deep dive into the learning algorithm. At this point, the environment has not been solved yet. But a plot of rewards per episode is showed to illustrate the learning process, along with a list of ideas on how to further improve my agent's performance. I hope to receive reviewers' feedback on these ideas so that I can incorporate a selected few in my next submission.

Environment

The Reacher environment is provided by DRLND's course website. In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of my agent is to maintain its position at the target location for as many time steps as possible. Figure 1 is a screenshot of the environment.

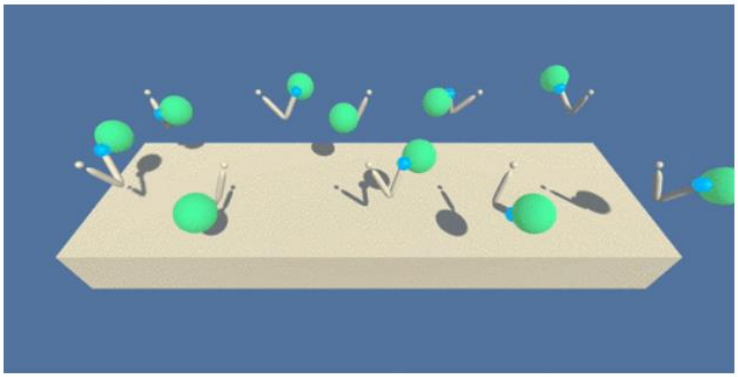

Figure 1. A screenshot of the Continuous Control (Reacher) environment.

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. Every entry in the action vector should be a number between -1 and 1.

For this project, two separate versions of the Unity environment are provided:

- The first version contains a single agent.
- The second version contains 20 identical agents, each with its own copy of the environment.

There are two options to solve the environment:

Option 1: Solve the First Version

The task is episodic, and in order to solve the environment, my agent must get an average score of +30 over 100 consecutive episodes.

Option 2: Solve the Second Version

The barrier for solving the second version of the environment is slightly different, to take into account the presence of many agents. In particular, my agents must get an average score of +30 (over 100 consecutive episodes, and over all agents).

Learning Algorithm

I used the DDPG algorithm [1] to solve this environment. DDPG is basically an actor-critic method. The actor network maps states to the continuous action space, while the critic network approximates the Q-value of state-action pair. The goal is to maximize the Q-value by adjusting the weights of actor and critic networks. Inspired by the success of deep Q-network (DQN) [2], updating critic network is also driven by minimizing the temporal-difference from one time-step to the next. The two significant improvements of DQN over traditional online Q-learning - namely experience replay and fixed target network – are adopted in the DDPG algorithm. A pseudocode from [1] is shown below for reference.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

I built two 2-layer feedforward neural networks (NN) for the actor and critic, respectively. Table 1 shows the hyperparameters of these two networks along with some other learning parameters in the DDPG algorithm.

Table 1. Hyperparameters of DDPG and other learning parameters

Parameter	Description	Value
State_size	Dimension of the environment	33
Action_size	Dimension of the action space	4
BUFFER_SIZE	Replay buffer size	1,000,000
BATCH_SIZE	Minibatch size	1024
GAMMA	Discount factor	0.99
TAU	For soft update of target parameters	1e-3
LR_ACTOR	Learning rate of actor network	5e-4
LR_CRITIC	Learning rate of critic network	1e-3
UPDATE_EVERY	How often (timesteps) to update the target network	100
UPDATE_RUNS	If update, how many times in a row	10
N_episodes	Total episodes in training	400
Max_t	Maximum time steps in each episode	1000
[state_size, fc1_units, fc2_units, action_size]	Units in each layer of the actor network	[33, 128, 64, 4]
[[state_size, action_size], fcs1_units, fc2_units, output_size]	Units in each layer of the critic network	[[33, 4], 64, 64, 1]

Result and Discussion

I leveraged the DDPG Python implementation from the course GitHub repository (ddpg-pendulum) for this project. Figure 2 below shows the score of each episode (in blue) as well as the moving-average across past 100 episodes (in red dash line). Note that these scores are averaged across 20 agents in the second version of environment. According to the project description, this environment is "solved" once the agent is able to receive an average reward (over 100 episodes, across all agents) of at least +30. At this point, we have not reached the goal yet.

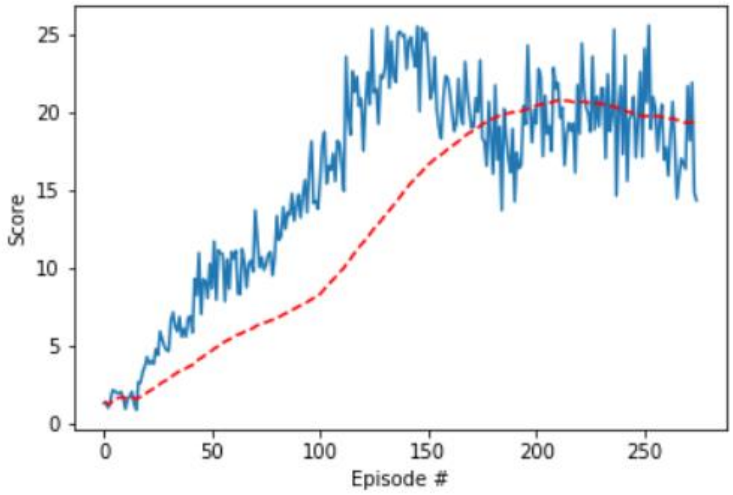

Figure 2. Averaged score over 20 agents (blue) and its 100-point moving average (red) during training using a DDPG algorithm.

However, I'd like to discuss a few crucial findings that helped me to get to this point. In addition, a specific question is asked for each finding that needs the reviewer's attention when providing comments to my submission.

1. **Batch Size**. By far, this is the parameter that dominates my agent's performance. I ended up using N=1024 for the reason of stability. The agent failed to learn anything if a much smaller batch (e.g. N = 128) was used. Even with N=512, the agent crashed after about 150 episodes after reaching 20+ for a short period of time. However, the relatively large batch size slowed down the training process quite a bit. It took me almost 24 hours to generate Figure 2 without access to GPU.
   [Question for Reviewer] What's your recommended batch size to balance performance stability and training efficiency?

2. **Frequency of Update**. I was suggested to update my networks 10 times every 20 time steps from the review of my last submission. I did try this, but it turned out that the updates were still too "frequent". The agent barely reached a score of +2 after 100 episodes. I eventually landed on a policy that updates my networks 10 times every 100 time steps.
   [Question for Reviewer] What's the general rule on the frequency of update?

3. **Network Size.** I used two 2-layer feedforward networks with each layer having no more than 128 units. I feel comfortable with my network size. However, the agent never reached a score of 30 in any of the episode it experienced. This makes me think that it was trapped in a local minima because of the error in function approximation. I'm not sure this is an overfitting or underfitting problem.
   [Question for Reviewer] What's your advice on the network size in order to achieve a relatively high score (+30)?

4. **Reward Shaping.** In my last submission, the reviewer asked about the purpose of ensuring a reward of +0.1 after each non-zero reward time step. I think it is necessary because the environment does NOT output +0.1 if I print out the reward. It's typically some value between 0.02 and 0.04. For example, in cell 9 of my Jupyter notebook, for the same episode, you will find the unadjusted and adjusted reward are 6.4 and 16.9, respectively. I'm not sure what is causing this problem.
   [Question for Reviewer] Do you agree this reward shaping is necessary? If not, how to explain the discrepancy?

Ideas for Future Work

I am expecting answers to questions listed in the last section. This would help me tweak my networks to achieve satisfactory performance still using the DDPG algorithm. Of course, I can explore some other policy gradient methods such as PPO and A3C. But I'd like to reserve them for later only if I'm not be able to solve the environment with my current framework.

Reference

[1] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. *Continuous control with deep reinforcement learning*. ICLR 2016

[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, 518 (7540):529–533, 2015.