

Deep Reinforcement Learning Nanodegree Project 2 - Continuous Control

KyoungSu Lee July 12, 2020

In this report, I have trained an agent that has continuous action spaces with Deep Deterministic Policy Gradient (DDPG) algorithms. The Reacher of Unity Environment, which is a double jointed arm that moves to target location, is used. I tested with 20 agents to collect experiences in parallel. The agent receives a reward of +0.1 for each step and the project goal was to achieve average scores +30 over 100 consecutive episodes over all agents. (Option 2)

Unity Reacher Agent Environment:

1. Number of agents: 20
2. Size of action for each agent: 4
 - A. Value between [-1,1]
3. Size of states for each agent: 33

I tested with multiple hidden layers (400,300), (256,128), (128,64), (64,32). I searched other papers and got these hidden layer sets.

Deep Deterministic Policy Gradient (DDPG) Algorithm

I trained the actor and the critic network with DDPG algorithm. Actor networks create policy network that takes state values as input parameter and returns the actions. Critics calculate Q values taking current states and actions that predicted from actor policy as inputs. Target networks are simply replicas of Actor, Critics were used to reduce the variance of the networks. I used Replay buffer to save experiences from all agents.

Critic loss - Mean Squared Error of $y - Q(s, a)$ where y is the expected return as seen by the Target network, and $Q(s, a)$ is action value predicted by the Critic network. y is a moving target that the critic model tries to achieve; we make this target stable by updating the Target model slowly.

Actor loss - This is computed using the mean of the value given by the Critic network for the actions taken by the Actor network. We seek to maximize this quantity.

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

Methods

I found many documents without Batch Normalization, the training results were the worst. So I used Batch Normalization for the Actor and the Critic. I tested with below hidden layers

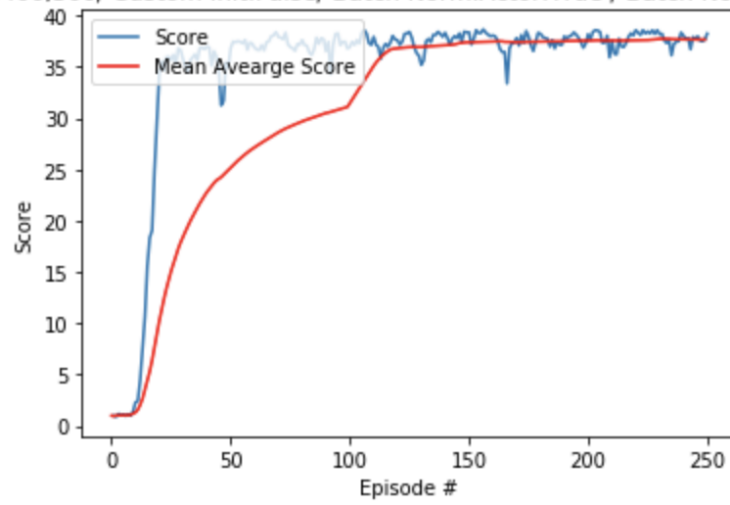
1. Hidden Layers: 400, 300
2. Hidden Layers: 256, 128
3. Hidden Layers: 128, 64
4. Hidden Layers: 64, 32

Results

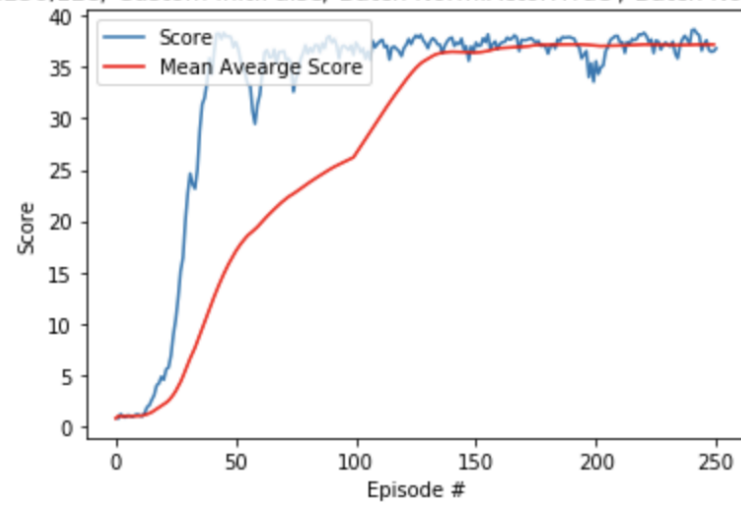
The result of agent trainings for DDPG algorithms are shown below. The agents solved the environment in 80 ~ 140 episodes. 80 episodes is enough for finding the goal, +30 scores in 400x300 Hidden Layers. And the Hidden Layers 400x300 was the best performance

hidden layers	mean_score	first_episode_achieved
400, 300	37.7503	85
256, 128	37.2035	111
128, 64	35.4912	126
64, 32	32.6857	140

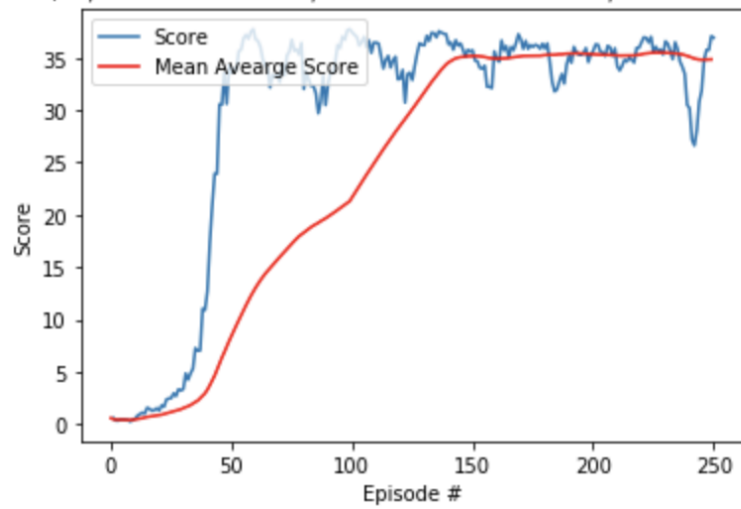
Layers:400,300/ Custom Init:False/ Batch Norm:Actor:True / Batch Norm.Critic:True



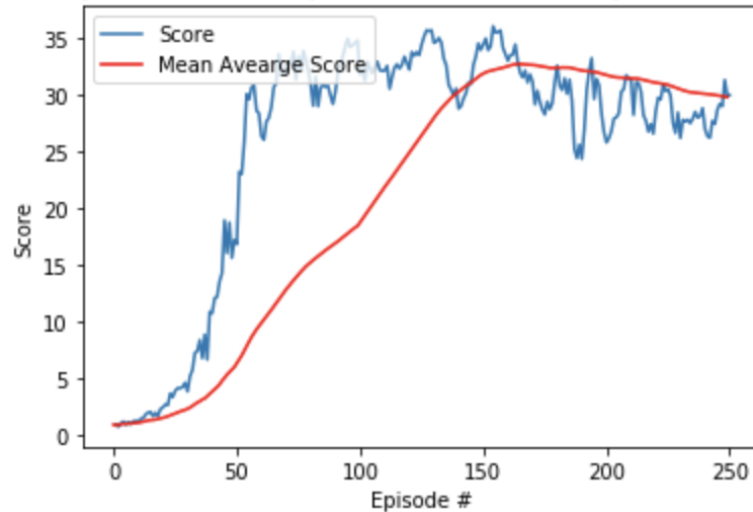
Layers:256,128/ Custom Init:False/ Batch Norm:Actor:True / Batch Norm.Critic:True



Layers:128,64/ Custom Init:False/ Batch Norm:Actor:True / Batch Norm.Critic:True



Layers:64,32/ Custom Init:False/ Batch Norm.Actor:True / Batch Norm.Critic:True



Future Works

At the end of train result, I found Keras Examples which shows more clean codes and optimized hyperparameters. And the training took almost 12 hours. I read an article about new version of ML-Agents v1.0 release 3 which reduce training time. Next time I'll try to train with this version.

Also, as the reviewer guided, Prioritised Experience Reply(PER)[4], is very interesting algorithm. In this project, a memory buffer is used to recycle older experiences of the agent in the environment and improve learning. This is called experience replay. The PER has important concept called Prioritising of experiences. Next time I'll try to train with this algorithm.

References

1. CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING, Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra Dueling Network Architectures for Deep Reinforcement Learning, Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas <https://arxiv.org/pdf/1509.02971.pdf> (<https://arxiv.org/pdf/1509.02971.pdf>)
2. Using Deep Reinforcement Learning for the Continuous Control of Robotic Arms, Winfried Löttsch, <https://arxiv.org/pdf/1810.06746.pdf> (<https://arxiv.org/pdf/1810.06746.pdf>)
3. Keras Examples https://keras.io/examples/rl/ddpg_pendulum/ (https://keras.io/examples/rl/ddpg_pendulum/)
4. Prioritised Experience Replay in Deep Q Learning, <https://adventuresinmachinelearning.com/prioritised-experience-replay/> (<https://adventuresinmachinelearning.com/prioritised-experience-replay/>)

In []: