Udacity Deep Reinforcement Learning

Project 2: Continuous Control

Jihye Seo

March 9, 2020

Introduction

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

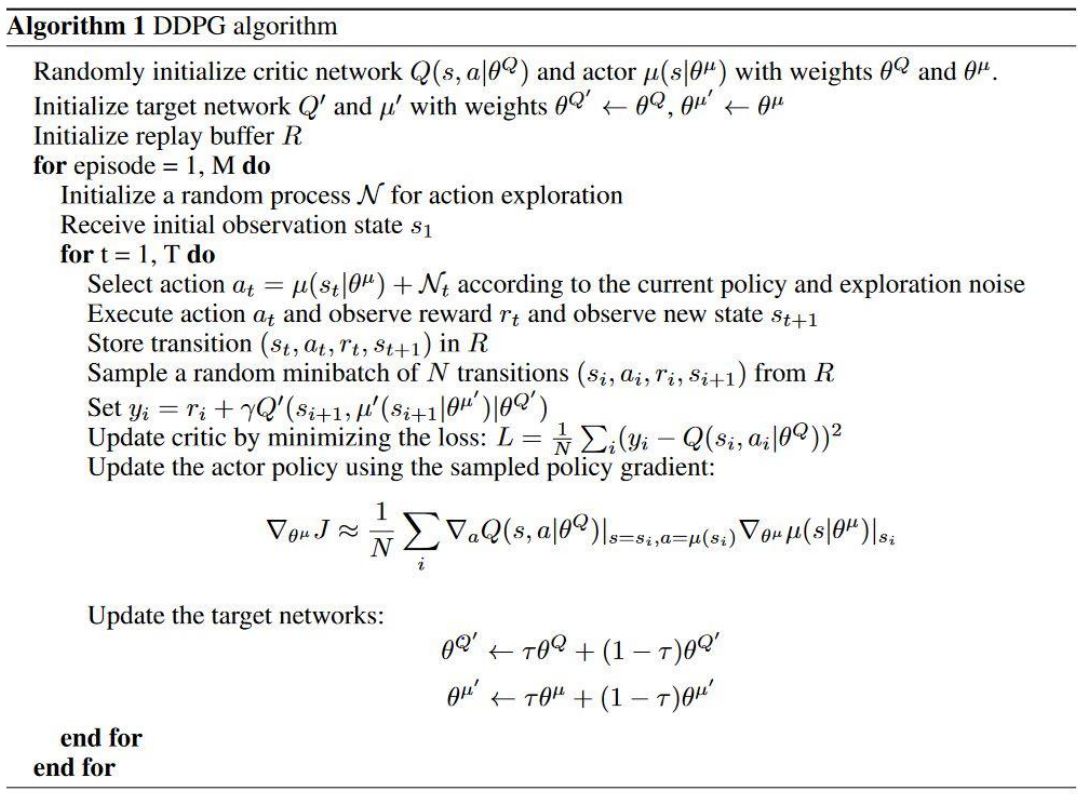
Unity Reacher Agent Environment:

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

I have started from the default setting that recommended in the DDPG paper. However, I had quite hard time to observing the agent getting trained even for 500 episodes. So I have tried different configurations as shown in methods section. Random Seed number even had big impact on my training. For example, only after chaining random seed 10 to 2, I was able to train network.

DDPG (Deep Deterministic Policy Gradient) Algorithm

In this project I used DDPG algorithm to train the actor and critic network. Actor networks creates policy network that takes states value as input parameter and returns the actions. Critics calculates Q values taking current states and actions that predicted from actor policy as inputs. Target networks simply replica of Actor, Critics were used to reduce the variance of the networks. Instead of fixing target for specific time interval, target slowly updates its network parameters at every learning step. Replay buffer was also used to save experiences from all agents. In my case, I have used 20 agents and updated the network every time step after collecting experiences from 20 agents each (20 experiences).

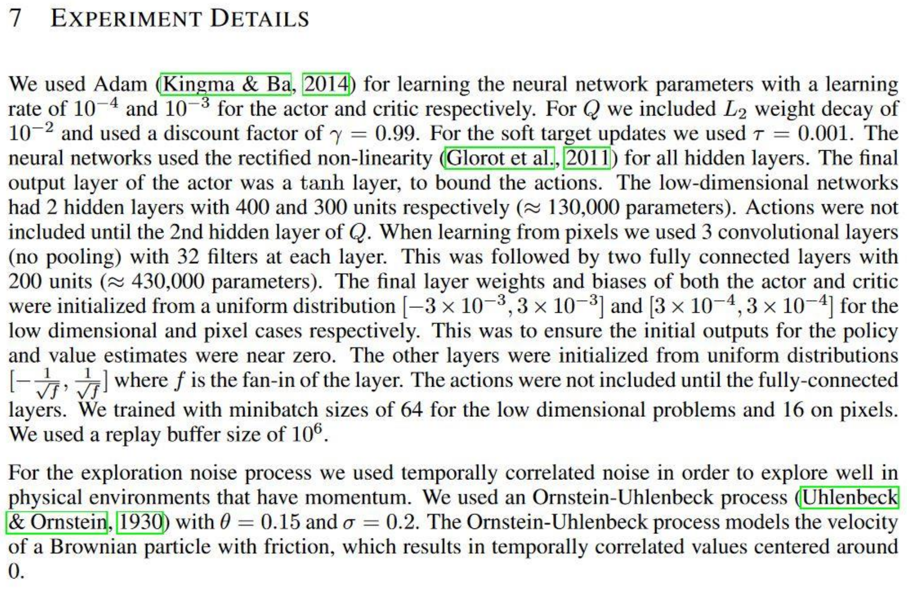


Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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

Using 20 agents, I tried the hyper parameter setting and Batch Normalization recommended in paper. I have changed some configuration and tested following combinations shown in table 1. Default setting means configuration from the DDPG paper which shown below.



7 Experiment Details

We used Adam (Kingma & Ba, 2014) for learning the neural network parameters with a learning rate of 10^{-4} and 10^{-3} for the actor and critic respectively. For Q we included L_2 weight decay of 10^{-2} and used a discount factor of $\gamma = 0.99$. For the soft target updates we used $\tau = 0.001$. The neural networks used the rectified non-linearity (Glorot et al., 2011) for all hidden layers. The final output layer of the actor was a tanh layer, to bound the actions. The low-dimensional networks had 2 hidden layers with 400 and 300 units respectively ($\approx$ 130,000 parameters). Actions were not included until the 2nd hidden layer of Q. When learning from pixels we used 3 convolutional layers (no pooling) with 32 filters at each layer. This was followed by two fully connected layers with 200 units ($\approx$ 430,000 parameters). The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ and $[3 \times 10^{-4}, 3 \times 10^{-4}]$ for the low dimensional and pixel cases respectively. This was to ensure the initial outputs for the policy and value estimates were near zero. The other layers were initialized from uniform distributions $[-\frac{1}{\sqrt{f}}, \frac{1}{\sqrt{f}}]$ where f is the fan-in of the layer. The actions were not included until the fully-connected layers. We trained with minibatch sizes of 64 for the low dimensional problems and 16 on pixels. We used a replay buffer size of 10^6.

For the exploration noise process we used temporally correlated noise in order to explore well in physical environments that have momentum. We used an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) with $\theta = 0.15$ and $\sigma = 0.2$. The Ornstein-Uhlenbeck process models the velocity of a Brownian particle with friction, which results in temporally correlated values centered around 0.

- Default Batch norm: State input and all layers of Actor network, All layers before Critic
- Default Hidden Layer network: 400,300
- Default Initialization:
  - Uniform distribution between [-1/sqrt(f), 1/sqrt(f)), where f is fan-in of the layer.
  - Final layer of Actor [-3x10^-3, 3x10^3],
  - Final Layer of Critic [-3x10^-4, 3x10^4]

-

- **Table 1. Testing conditions**

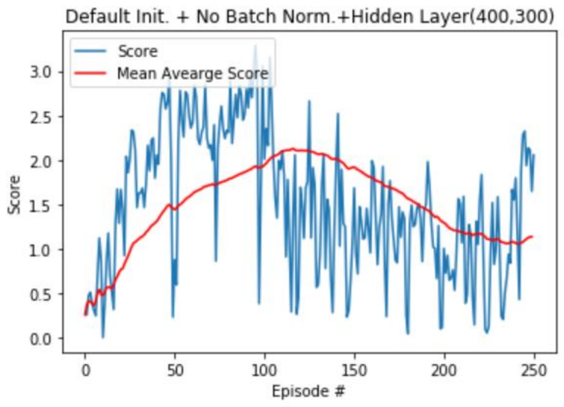
	Weight Initialization		Batch Normalization			Hidden Layer Nodes	
	Default	Xavier	No Batch Norm	Default Batch norm	Batch norm on states inputs only (both A/C)	400,300	256,128
0	Yes		Yes			Yes	
1	Yes		Yes				Yes
2	Yes			Yes			Yes
3		Yes	Yes				Yes
4		Yes		Yes			Yes
5		Yes			Yes		Yes
6		Yes			Yes	Yes	
7	Yes			Yes		Yes	

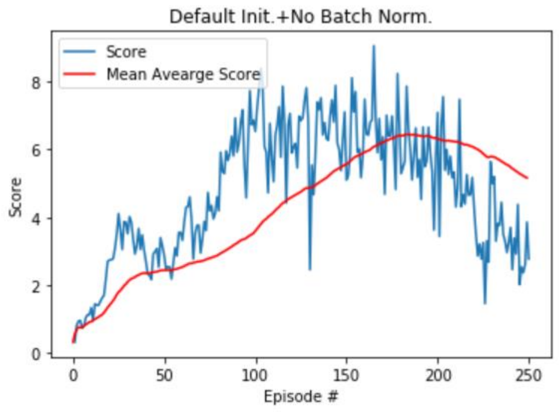
Results

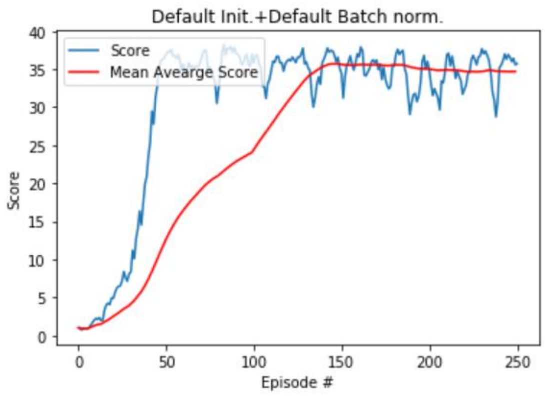
This is the final results. Batch normalization seems to have largest impact on the agent learning while number of hidden layer nodes didn't have notable impact. While default setting showed good results, the Xavier initialization seem to give stable results after it achieves the goal.

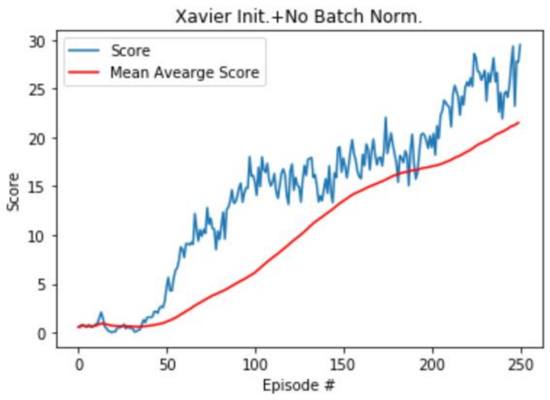
	Weight Initialization		Batch Normalization			Hidden Layer		Result		
	Default	Xavier	No Batch Norm	Default Batch Norm	Batch Norm on States inputs	400, 300	256, 128	Max Score	Max Mean Score for 100 episodes	First goal Achieved Episode
0	Yes		Yes			Yes		3.2875	2.128665	-1
1	Yes		Yes				Yes	9.056	6.44399	-1
2	Yes			Yes			Yes	38.1865	35.69906	119
3		Yes	Yes				Yes	29.481	21.50603	-1
4		Yes		Yes			Yes	37.62	34.37656	178
5		Yes			Yes		Yes	37.421	35.88789	171

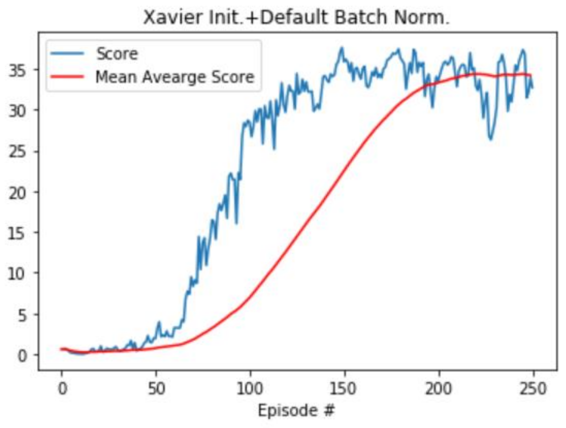
6		Yes			Yes	Yes		37.3235	35.63318	135
7	Yes			Yes		Yes		38.9895	37.05086	118

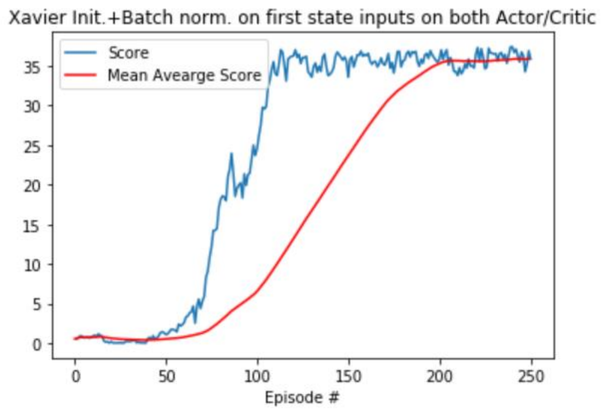


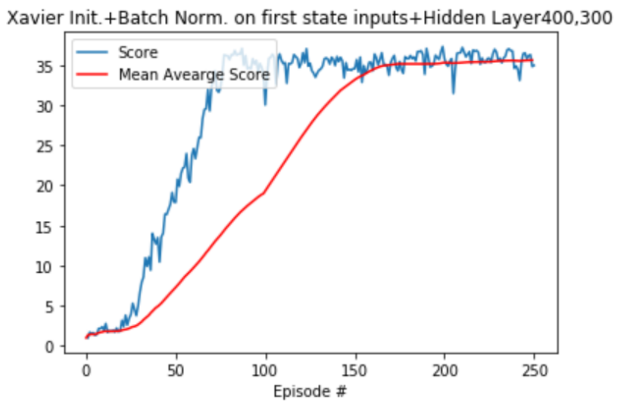


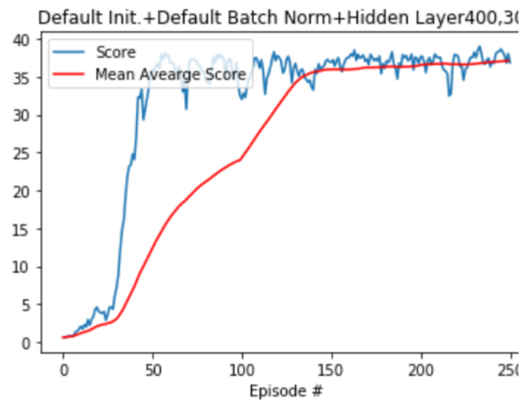






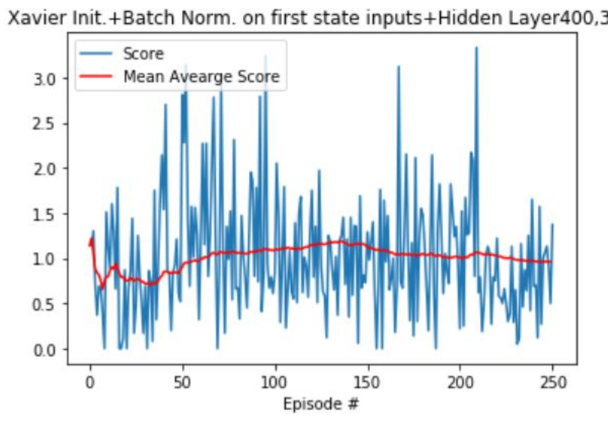


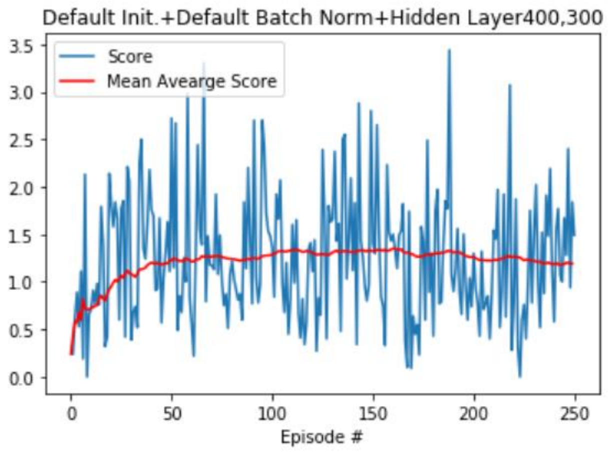




Future Work

Due to time limit I wasn’t able to finish up another experiments with other algorithms such as A3C, PPO. I have also tried single agent version. But unfortunately, some best results from above combination didn’t worked in the single agent case as shown below. Also Random seed number was big issue for me in 20 agent cases and only after changing random seed number 10 to 2 I was able to solve issues. This would be more area for research for me.


Xavier Init.+Batch Norm. on first state inputs+Hidden Layer400,3
Score
Mean Avearge Score
Episode #


Default Init.+Default Batch Norm+Hidden Layer400,300
Score
Mean Avearge Score
Episode #