



# Continuous Control

01.12.2019

—

Anu Pradhan

## Overview

To solve the continuous control problem (Version 1), I used Deep Deterministic Policy Gradient (DDPG) algorithm invented by Lillicrap et al. 2016. I modified the DDPG code used to solve the pingpong problem. Overall, this project is particularly challenging in terms of identifying a good set of hyperparameters. The DDPG algorithm seems to be sensitive to the hyper-parameters. After spending more than a week experimenting with the hyper-parameters and modifying the code, I am able to achieve the reward of 30+.

## Hyperparameters

I ended up using the following values for the hyper-parameters

```
BUFFER_SIZE = int(1e6) # replay buffer size
BATCH_SIZE = 256 # minibatch size
GAMMA = 0.99 # discount factor
TAU = 1e-3 # for soft update of target parameters
LR_ACTOR = 1e-4 # learning rate of the actor
LR_CRITIC = 1e-3 # learning rate of the critic
WEIGHT_DECAY = 0.0000 # L2 weight decay
```

To tune the above hyperparameters, I looked into the Supplementary Information section of the Lillicrap et al. paper. I ended up using most of the values suggested by the paper. Though I tried to tweak some of the values initially, I was not able to get good results. Since each experiment takes a couple of hours, I was not able to do a thorough job in terms of my experiments.

## Neural Network Architecture

Actor and Critic Networks are implemented as a feed-forward neural network. Each network consists of 2 hidden layers with 400 and 300 units respectively. Batch Normalization is performed at each hidden layer, and Batch Normalization was able to reduce the number of iterations needed to

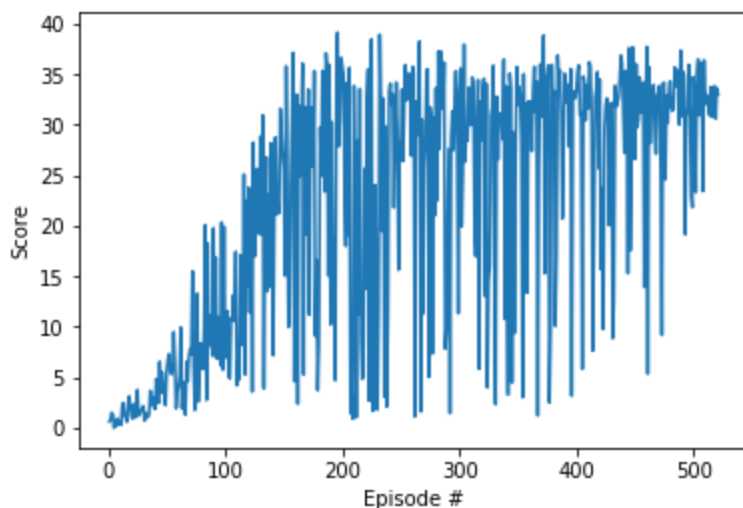
train the networks. RELU activation is used at each hidden layer. For the output layer, tanh activation is used for the actor network, and no activation function is used for the critic network.

Based on the suggestion made in the Benchmark Implementation (Udacity), I used gradient clipping as shown below.

```
self.critic_optimizer.zero_grad()
critic_loss.backward()
torch.nn.utils.clip_grad_norm(self.critic_local.parameters(), 1)
self.critic_optimizer.step()
```

## Overall Performance

To achieve the reward of 30+, I need to run around 500 episodes as shown in the plot. The plot is not smooth as shown in the Benchmark Implementation (Udacity). I am not sure about the reason behind this.




To speedup the learning, I tried to update the network 10 times for every time step as suggested in the Benchmark Implementation page. This helped significantly to speed up the learning process.

## Future Improvements

### Prioritized Experience Replay

Instead of randomly sampling the (S A R S') tuples, we can prioritize the sampling process based on the TD error. This has shown to improve the learning process.

### Additional Experiments on Hyperparameters



Because of computational limitations and time constraints, I was not able to do a thorough job in terms of tuning the hyperparameters. Hypertuning could help to further improve the performance.

**Distributed Distributional Deterministic Policy Gradient**

Barth-Maroon et al. (2018) extended DDPG within a distributed framework and they showed that this could improve the standard DDPG. Using the improved DDPG could be another future improvement.