Udacity DRL Nano Degree

Project Continuous Control

Fabrice R. Noreils

Introduction

Let us first describe the set-up which consists in a double-jointed arm which can move to target locations.
Here are the main important facts :

- Goal: The agents must move it's hand to the goal location, and keep it there.
- Agent Reward Function (independent): +0.1 Each step agent's hand is in goal location.
- Brains: One Brain with the following observation/action space.
    - Vector Observation space: 33 variables corresponding to position, rotation, velocity, and angular velocities of the two arm Rigidbodies.
    - Vector Action space: (Continuous) Size of 4, corresponding to torque applicable to two joints.
    - Visual Observations: None.
- Reset Parameters: Two, corresponding to goal size, and goal movement speed.
- Benchmark Mean Reward: 30

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

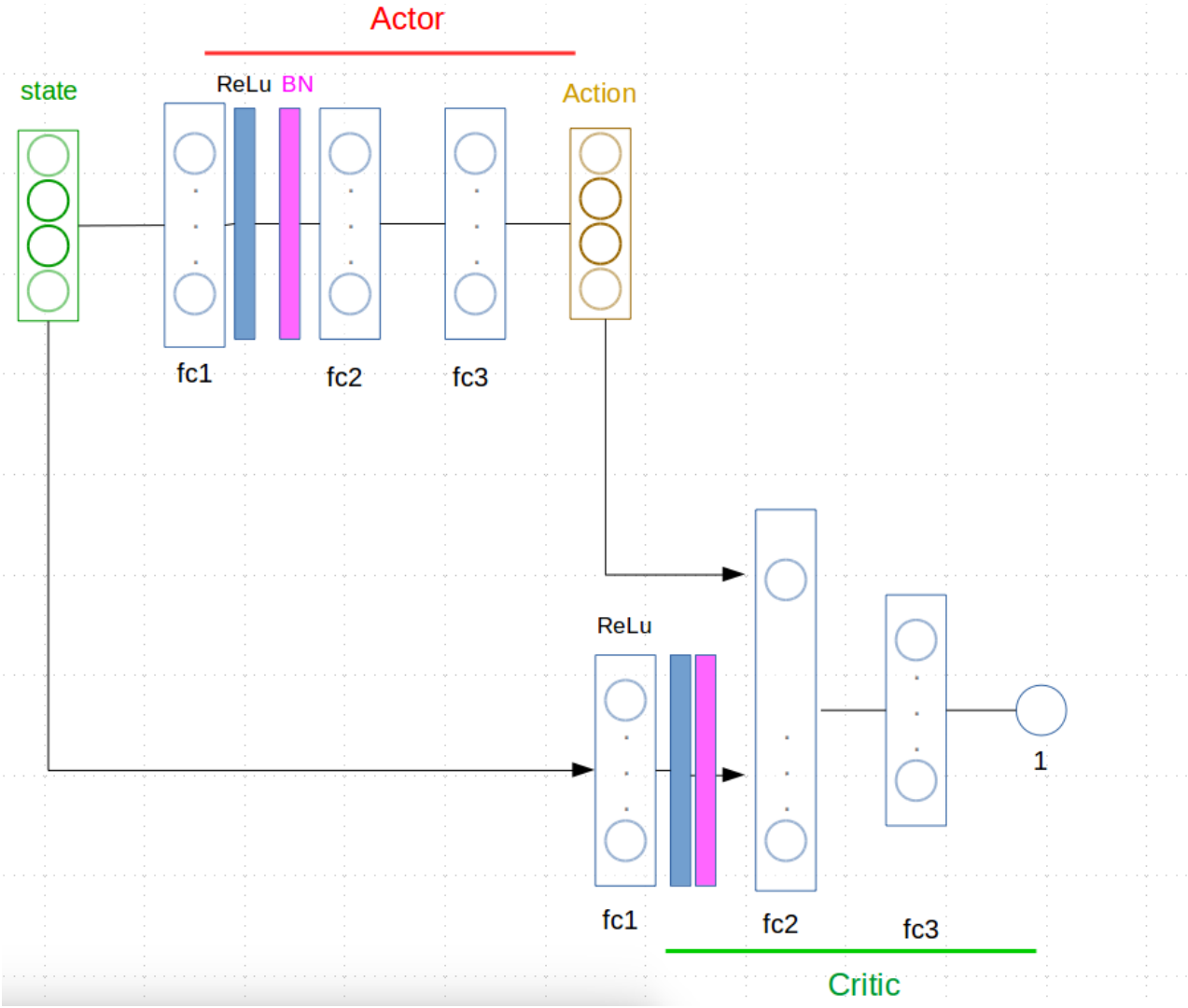
I decided to implement DDPG for one agent only.

Network Architectures

The architecture is presented on the Picture below. This picture is interesting because it shows how the actor and the critic are coupled.

It consists in 3 fully connected layers with ReLu activations (after the first one) for both the actor and the critic. However, following the recommandation of the authors [1], the Critic network takes as a first input the state Tensor and then concatenates the ouptut with the action Tensor before going through the second fully connected layer.

I tried different units combinations : 64/64, 128/128 and 256/256 units for the two hidden layers and finally used 128/128 in my final setup because it provided the best result.



Overview of the DDPG networks for the Actor and the Critic.

However, in order to train the DDPG agent, I did two main improvements :

- Batch normalization (BN) after the first hidden layer for the actor network an the Critic networks. Then, the training started to get somewhere. I tried to add a second BN after the second hidden layer but the result was worse than with one BN layer only ;
- Hard copy of the weights at the initialization of the agent, as it was suggested by the authors as well [1].

Another trick which important to mention is that instead of computed the gradient below to update the actor policy :

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

We use this equation :

$Policy\ loss = \nabla_{\theta^\mu} Q(s, \mu(s|\theta^\mu)|\theta^Q)$ which is coded in pytorch like this in the *learn* function:

```
actions_pred = self.actor_local(states)
actor_loss = -self.critic_local(states, actions_pred).mean()
# Minimize the loss
self.actor_optimizer.zero_grad()
actor_loss.backward()
self.actor_optimizer.step()
```

I scratched my head for a while in order to understand this equation and after talking with some (wise) people on pyTorch Forum, I got the explanation : iSince Q approximates our return, we want to maximize :

$$E_{a\sim\mu(s)} = E_s[Q(s,a)] = E_s[Q(s,\mu(s))] \qquad (1)$$

So we take the derivative :

$$E_s[\mu'(s) Q'(s,\mu(s))] \qquad (2)$$

and do a gradient descent with respect to the parameters of μ .

However, with an automatic derivation tool like pytorch, we just take the first equation as a loss to maximize (wrt mu's parameters).

Parameters

I tried different batchsizes : 64, 128 and 256. 64 did not work, 128 worked just fine and 256 did not bring significant improvements with respect to the 128 value.

I did not change the learning rate of both the actor and the critic, I kept 2e-4. Although my feeling was that a faster learning rate such as 0.001 should work as well.

I set up the weight decay to 0 as I saw that other students mentioned it and it did improve the results.

I did not change Gamma nor Tau (soft update)

I also experimented with a few different random seeds for the agent but I did not see any major differences in convergence behavior.

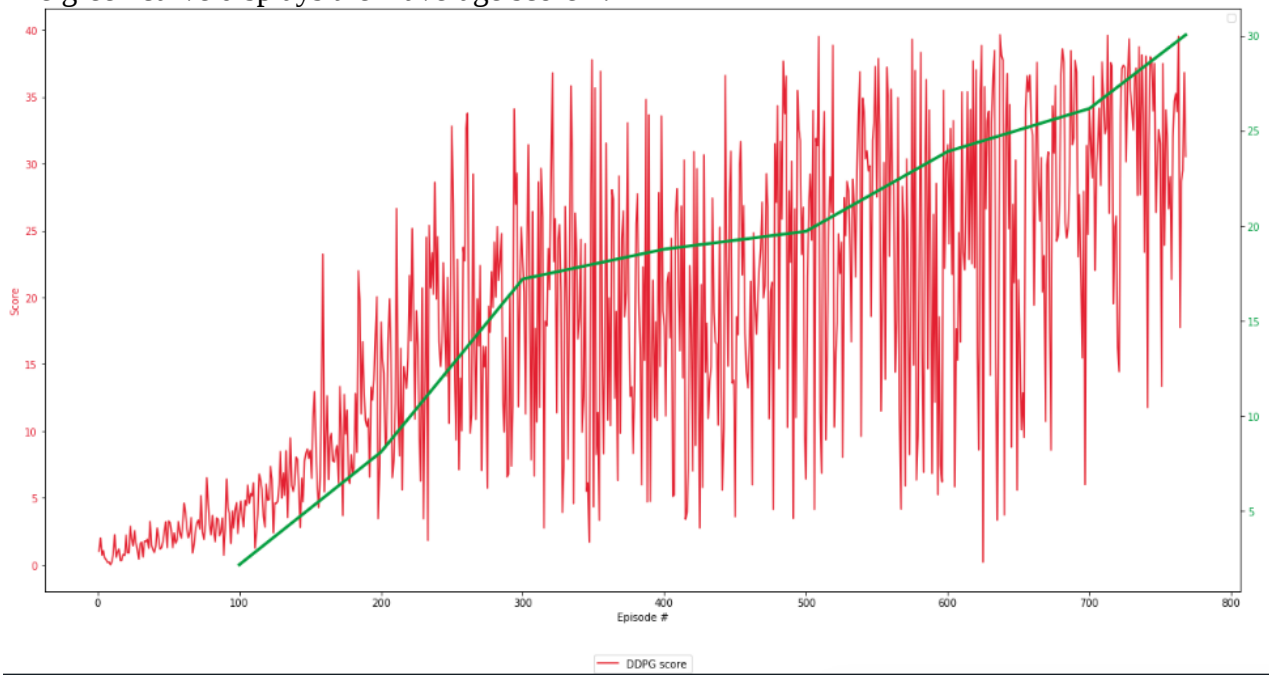
To summarize :

BUFFER_SIZE # replay buffer size	int(1e5)
BATCH_SIZE # minibatch size	128
GAMMA # discount factor	0.99
TAU # for soft update of target parameters	1e-3
LR_ACTOR # learning rate of the actor	2e-4
LR_CRITIC # learning rate of the critic	2e-4
WEIGHT_DECAY # L2 weight decay	0
FC1_UNITS/FC2_UNITS #dim fc layers	128

Results.

As it is displayed on the picture below, the environment could be solved in 768 episodes for a single agent.

The green curve displays the « average score ».



Further work.

I am thinking about introducing a prioritized replay buffer that may provide some improvement in terms of performance.

[1] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *CoRR, abs/1509.02971*.