Udacity - Continuous Control

Student's name: Paulo Cotta

Abstract

I apologize for the English, but it is not my native language and if you have any problems with writing, please feel free to point out the errors.

It is best to apply DDPG (Deep Deterministic Policy Gradients) on the Deep Q Network (DQN). Because the DDPG works in the continuous state and in the action space, while the DQN is intended to solve the discrete action space problems, the Critical Value Model calculates the Q values for any pair (state, action). Thereafter, the gradient of this Q value is calculated in relation to the corresponding action vector which is then fed into the training of the Actor Policy Model. A DDPG consists of two networks: an actor and a critic.

Here are the parameters that were used:

- BUFFER_SIZE = int(1e6)
- BATCH_SIZE = 128
- GAMMA = 0.99
- TAU = 1e-3
- LR_ACTOR = 1e-4
- LR_CRITIC = 1e-4
- WEIGHT_DECAY = 0.0
- N_LEARN_UPDATES = 10
- N_TIME_STEPS = 20

Here is the configuration that was used in the model inside the jupyter notebook for the DDPG function:

- n_episodes=300
- print_every=1000

Object Model and problems

Initially I had problems with the processing time and learning of Deep Deterministic Policy Gradients and the agents. Below is the table of executions:

Event	Average Score	Hours	Episodies	print_every
1	1.43	9	100	1000
2	4.73	12	100	1000
3	5.05.33	8	100	800
4	5.36.04	16	130	800
5	14.877	18	256	800
6	30.52	17	133	1000

Load state was used to reload the weights that were recorded in Workspace, it is worth remembering that I changed the function because I had no more GPU hours and I still have the last project to finish.

In this project, an agent (or several similar agents) aims to follow a goal. A +1 reward is provided for each step where the agent's hand is at the goal location. So, your agent's goal is to maintain your position at the destination for as many steps as possible.

Model Architecture

During a stage, the actor is used to estimate the best action, that is, argmaxaQ (s, a); the critic then uses this as in a DDQN to evaluate the optimal stock value function.

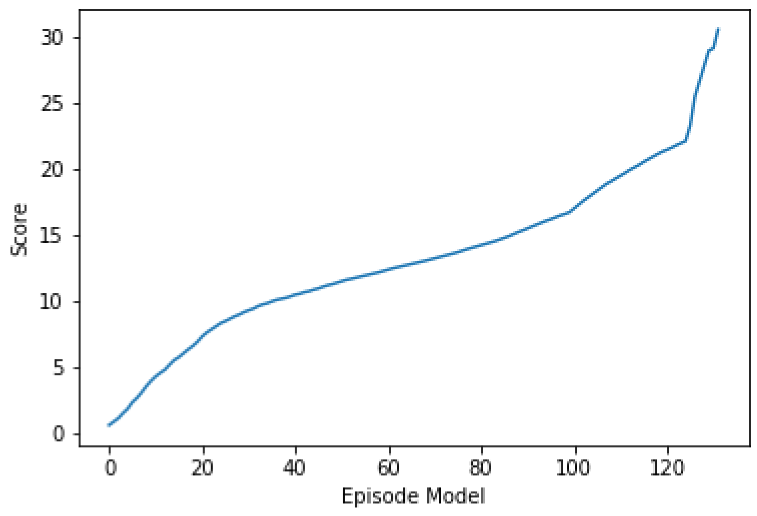
Both the actor and the critic are composed of two networks. On the local network and a destination network. This is for computing reasons: during backpropagation, if the same model was used to calculate the target value and the forecast, it would lead to computational difficulties.

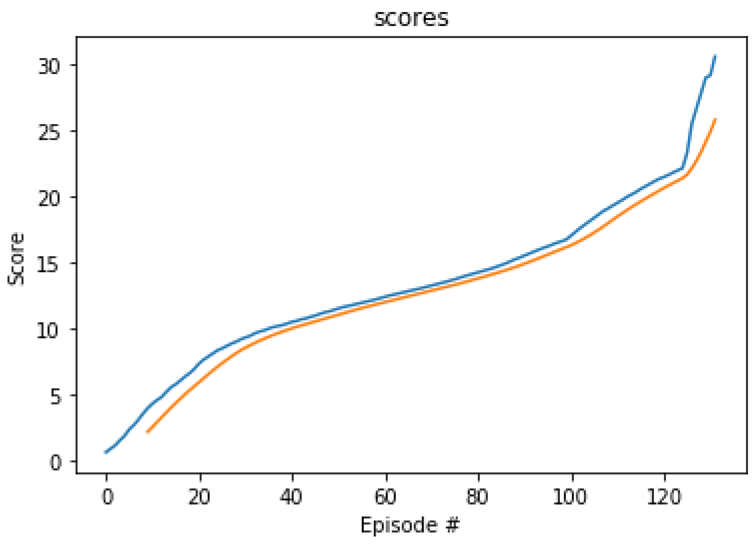
During training, the actor is updated by applying the chain rule to the expected return from the beginning of the distribution. The critic is updated as in Q-learning, that is, it compares the expected return of the current state with the sum of the reward of the chosen action + the expected return of the next state.

In this work, the ddpg.py and model.py files.

Final result - Conclusion

The work was very challenging and there were moments that I thought the agents were not going to learn. An important point is in the charts below:





Future Works

I'd like to try out algorithms like PPO, A3C, and D4PG that use multiple (non-interacting, parallel) copies of the same agent to distribute the task of gathering experience.

I'd like to try out dropouts, various weight initialization and further hyper-parameters like cost functions that may yield better results.

References

https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df
Book - Reinforcement Learning, Sutton