

Continuous Reinforcement Learning Problem Report

In this exercise, I used a Deterministic Deep Policy Gradient (DDPG) Agent to control a unity environment brain, that in turns controls an arm agent that must touch and object at all time. In this report, I documented the characteristics of the environment, the method used to solve the environment control problem, the result of the training process, and what can be done to improve it.

The Environment

In this environment, a double-jointed arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

The Agent

The agent is a version of a DDPG (<https://arxiv.org/pdf/1509.02971>) that uses two deep neural network the actor represents the agent policy, the critic calculates the q_value of the actions taken by the actor.

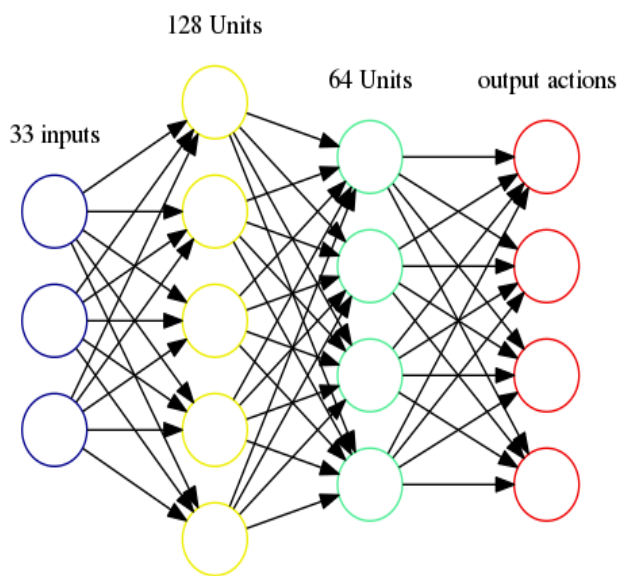
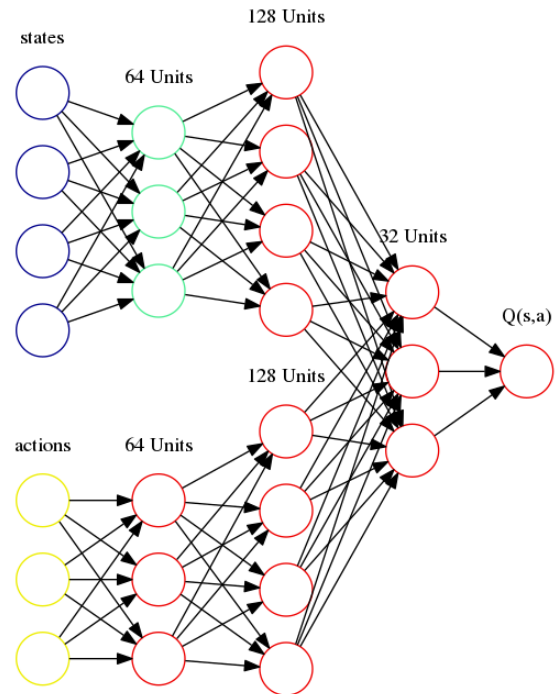


Illustration 1: Actor Network (left),



Critic Network (Righth)

The main idea behind the DDPG agents is the use on the Actor to determine directly the actions and their magnitude, that the agent has to apply in any given state. The role of the critic is used the action taken by the actor on a particular state and calculate the value of that action on that state.

The Optimization

The Critic is optimized by taken the action produced by the actor, the state in which it took the action, then use the result next state and the actor again we collect (using the same actor network) the action that applies to the next state. The resulting tuple: state, action, next_state and next_action, is used with the Bellman equation ([https://en.wikipedia.org/wiki/Bellman equation](https://en.wikipedia.org/wiki/Bellman_equation)) to calculate an estimate value of the q_value of the next state. We use the difference of the q_value calculated using the Bellman equation and the q_value produced by the critic as the loss or error signal of the critic, in turn we used that loss to optimized the critic.

The Actor optimization process is based on the David Silver “Deterministic Policy Gradient” paper (<http://proceedings.mlr.press/v32/silver14.pdf>). The paper explains that the error signal of the Actor or agent policy function is: the sum of the negative of the gradient of the Critic q_values with respect to

the actions, plus the the gradients of the Actor actions with respect to its model parameters. Using this calculation as an error signal we optimize the Actor or policy model.

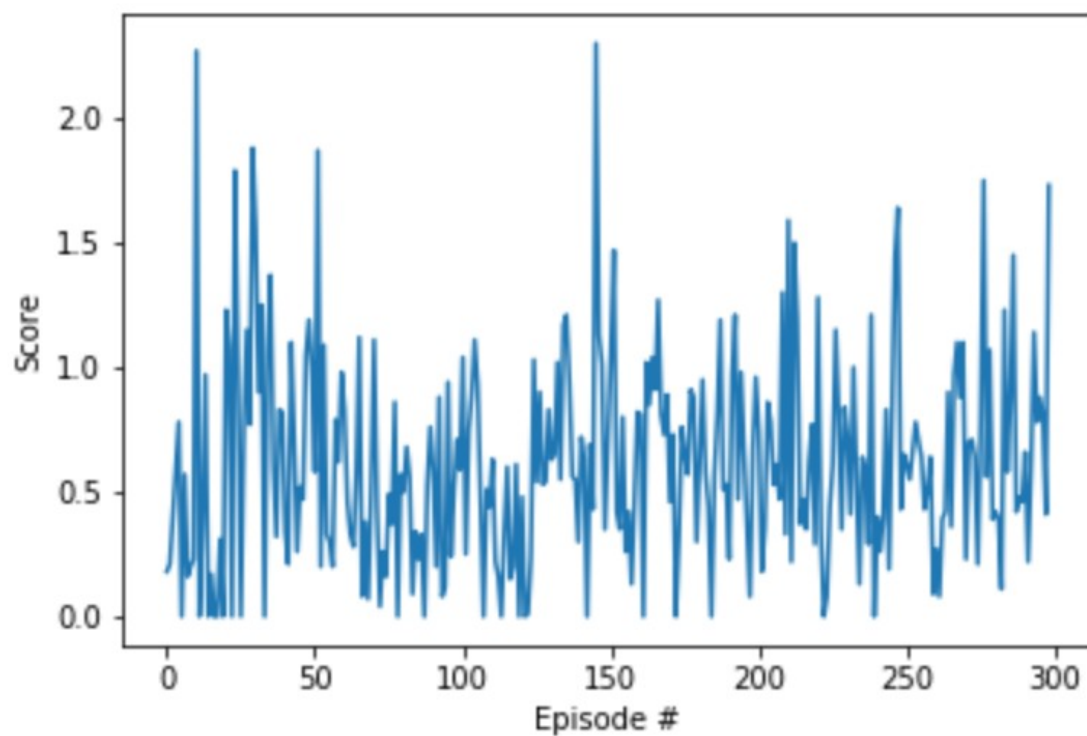
The parameters.

The replay buffer size was 1000; the learning batch consisted of 32 samples. I used a discount factor of 0.99, to perform a soft update of the target weights (see the fix q-targets weight documentation in the dqn URL) I used a TAU factor of 0,001, my learning rate was of 0.005, and I performed weighs update every other 10 time step, once I had at least 32 samples in the replay buffer.

The network was created using Pytorch.

The result of the training

After 300 episodes the model learns very slowly but shows a tendency of increasing the score (see screen shot below)



Improvements

Even if the agent was training to an acceptable level, the following improvement could be made to optimize the agent behavior and make it converge to the option q^* value in less training episodes.

Double Deep Q-Network

The problem with the dqn agent is that it tends to overestimate the q values, this is because it uses the q value if the actions that maximize the q estimate function to determine the real q value, this produces a tendency toward given values to actions higher than the real ones. The solution to this problem is the use of two sets of Q-target networks weights, one is used to select the action and the other set of weights is used to select the value of that action.

Prioritize Experience Replay

The main idea behind this improvement is that during the selection of the experiences to be used in the learning process, we select these using a random process, with this we run the risk of selecting the same set of samples or even worst not selecting valuable experiences. The improvement consists of using a parameter to weigh the importance of the experience base on its value. The value we can use to prioritize the selection of important experiences can be the error term obtained when we are performing the weigh optimization, using this value we can calculate the probability of the sample selection, the higher the experience error term the highest its probability is of been selected.