Report Continuous Control

Author: Dorzhi M.

Approach description

We apply DDPG to solve a continuous problem where a double-jointed arm can move to target locations. The approach we apply is based on '[Continuous control with deep reinforcement learning](#)' (Lillicrap et al.).

This problem is a typical reinforcement learning problem. We have an agent and an environment which responds to the agent's actions, and the agent receives rewards after each action.

Reinforcement learning problems can be addressed using value based and policy based methods. Unlike DQN, where we try to learn a value function of the action state space, policy based methods learn directly from the observation space by mapping states to actions using the gradient ascent.

One limitation of DQN is that it works only for discrete action space. DDPG combines actor-critic approach and the insights from DQN for tackling continuous problems. In DDPG we have two neural networks, one neural network, the 'Actor', is used to approximate the optimal policy deterministically, while the other network, the 'Critic', learns to evaluate the action value using Bellman equation as in DQN.

As DQN, DDPG utilizes a replay buffer to reduce bias from correlation between sequential samples. What is more, in DDPG, unlike DQN, target networks (both actor and critic) are updated using a "soft" update, which slowly mixes-in the regular network weights to the target network weights.

We solve single agent version of the environment. 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, meaning that the action space is not discrete.

Both actor and critic are fully connected neural network with 2 hidden layers. The following neural net is used for parametrizing Q function:

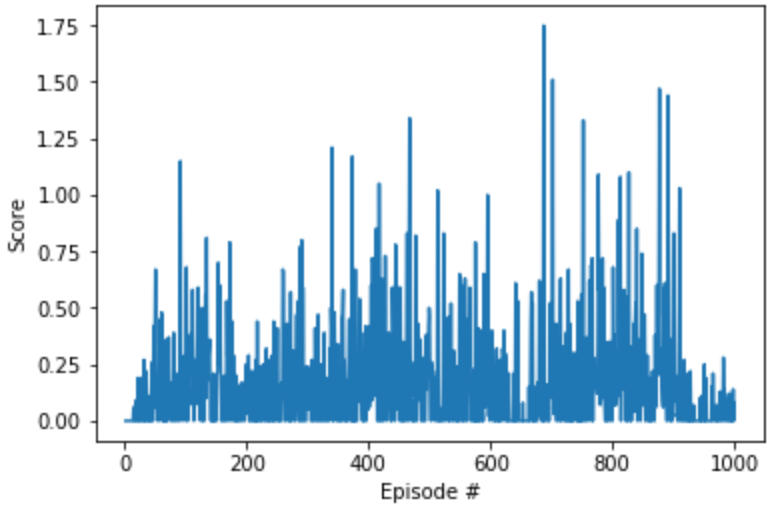
- 33-dimension input layer
- 1st hidden layer with dimension of 400, 2nd hidden layer with 300 dimensions
- 4-dimension output layer

In order to solve the environment, the agent must get an average score of +30 over 100 consecutive episodes.

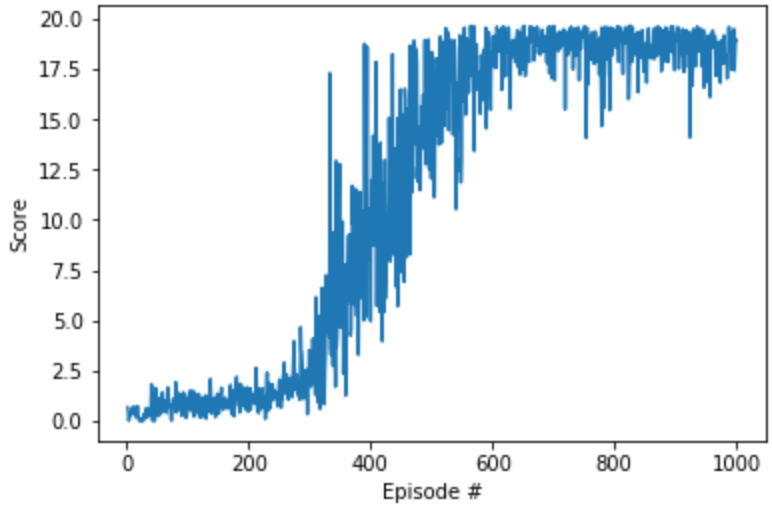
We set following hyperparameters:

- maximum number of training episodes (by default 1500)
- maximum number of timesteps per episode (by default 1500)
- replay buffer size (by default 1e5)
- minibatch size (by default 1024)
- discount factor gamma (by default 0.99)
- soft update of target parameters tau (by default 1e-3)
- learning rate of the actor (by default 1e-4)
- learning rate of the critic (by default 1e-3)
- L2 weight decay (by default 0)
- update frequency (by default 20)

At first, all the hyperparameters were taken from the example project, except for update frequency, which was taken from the recommendations to the project implementation. We got pretty bad performance.

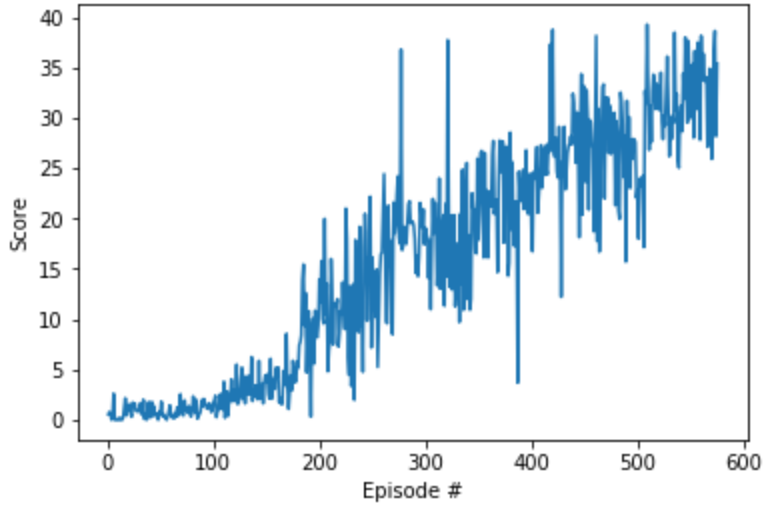


Then we increased the minibatch size to 512 and saw an increase in performance, however this increase was not enough to solve the problem.



Agent Performance

Finally, we increased the batch size to 1024 and maximum number of timesteps per episode to 1500. We were able to solve the problem in 476 episodes.



Ideas for future

As next steps, we can improve the performance of the agent by using prioritized experience replay, as in here. Some of the transitions are more important for the agent performance (those with higher error), and we can sample those transitions with higher probability, potentially speeding up the training process.