Project 2 Report

This report provides a description of the implementation of a deep reinforcement learning which is trained to move a double-jointed arm to target locations in an environment provided by a unity ml-agent called 'Reacher'.

The code is written in Python and PyTorch.

Learning algorithm

The learning algorithm used for the solution is based on the Deep Deterministic Policy Gradient (DDPG) agent. This includes a replay buffer implemented as a deque data structure.

The DDPG agent is an Actor-Critic which uses two Neural Networks, one for the actor and one for the critic. The actor is the policy function approximator which provides the critic with the best action vector to take at each time step. The critic is the action value function approximator and its aim it to approximate the optimal action value function and therefore the optimal total discounted reward for each state of the MDP.

The actor and the critic are two similar neural networks (NNs) characterised by two fully connected layers. The actor output layer has a number of nodes equivalent to the number of actions which the agent can take to interact with the environment. For the Reacher environment actions must belong to the interval [-1,1], therefore we apply a '*tanh'* function to the actor output layer before returning from the forward method of the actor class.

The critic output size is one as the critic network maps state action pairs to corresponding Q-values.

The algorithm uses the following hyperparameters:

- **fcs1_units** = 400: the number of units in the first fully connected layer of the Critic.
- **fc1_units** = 400: the number of units in the first fully connected layer of the Actor.
- **fc2_units** = 300: the number of units in the second fully connected layer of both Actor and Critic NNs.
- **BUFFER_SIZE** = 1e6: Size of the Replay Buffer.
- **BATCH_SIZE** = 1024: Number of inputs processed per batch when running Stochastic Gradient Descent.
- **GAMMA** = 0.99: Discount factor of the Q-Learning algorithm
- **TAU** = 1e-3: used to perform soft updates of the target networks' parameters
- **LR_ACTOR** = 1e-3: learning rate provided to the Adam optimiser for the Actor
- **LR_CRITIC** = 1e-3: learning rate provided to the Adam optimiser for the Critic
- **UPDATE_EVERY** = 4: parameter used to decide how often to update the network.
- **WEIGHT_DECAY** = 0: weight decay for the Adam optimiser.
- **UPDATE_EVERY =** 20.0**:** number of time steps we wait between consecutive networks' updates.

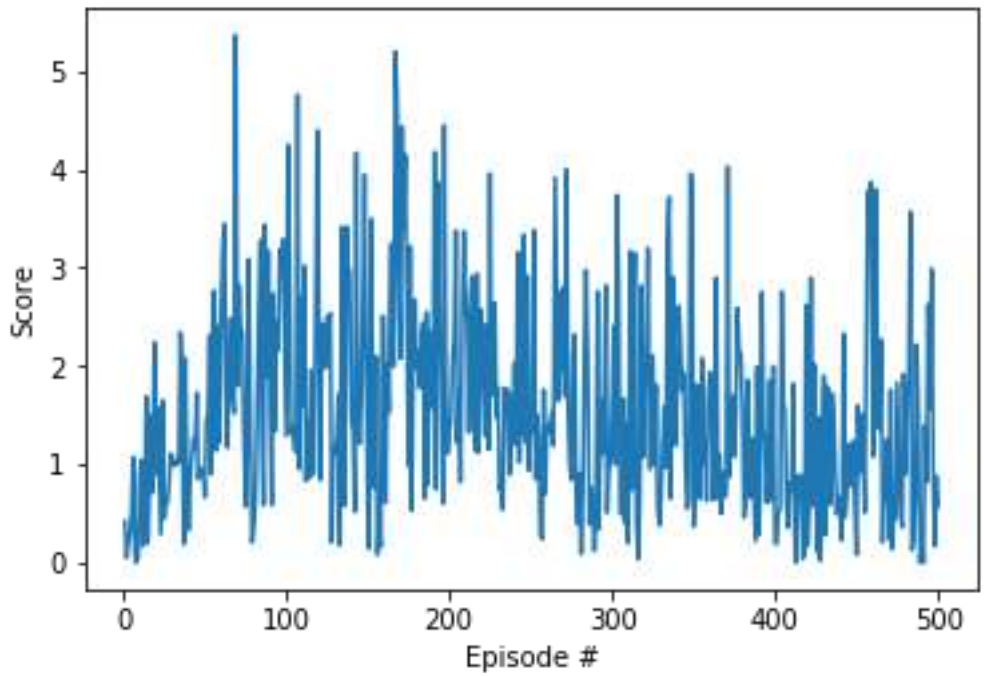
Plot of Rewards

I made multiple attempts at resolving this environment and it was not straightforward.

The final version is the result of lots of failures on the way!

I took as a baseline the model and agent structure which allowed me to get good results for the Open AI Gym pendulum environment.

I had learned from the literature that DDPG is quite sensitive to the frequency you update the actor and critic networks. I therefore started slightly modifying the baseline by reducing the number of networks' updates to one every four, time steps. As shown in the plot below this did not provide great results.

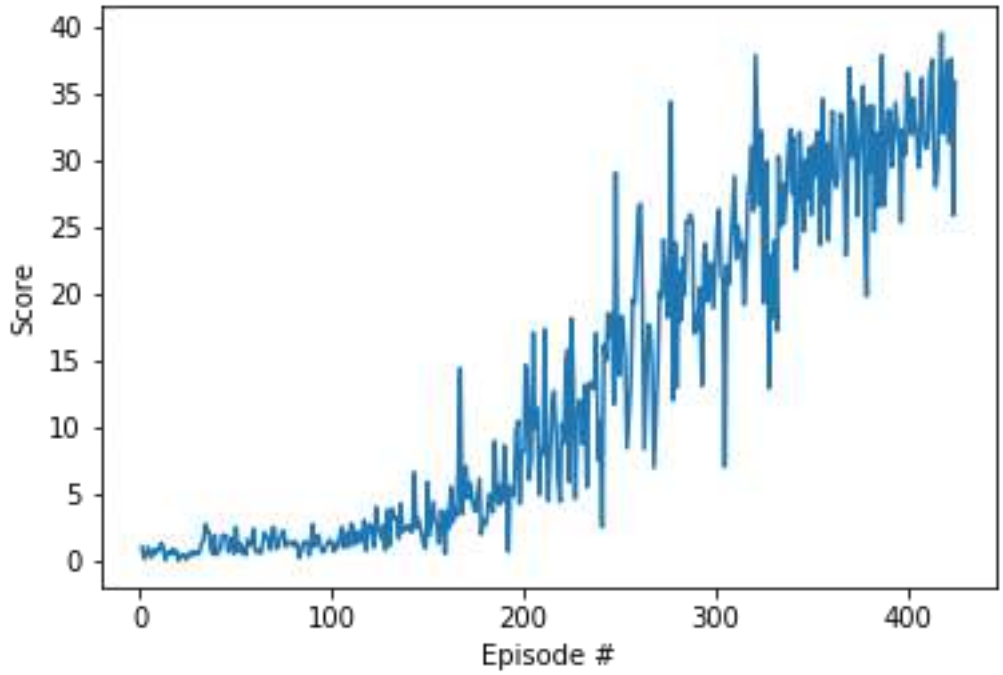


The agent was struggling to get episodic rewards higher than 5 and it was most of the time stuck just above 1!

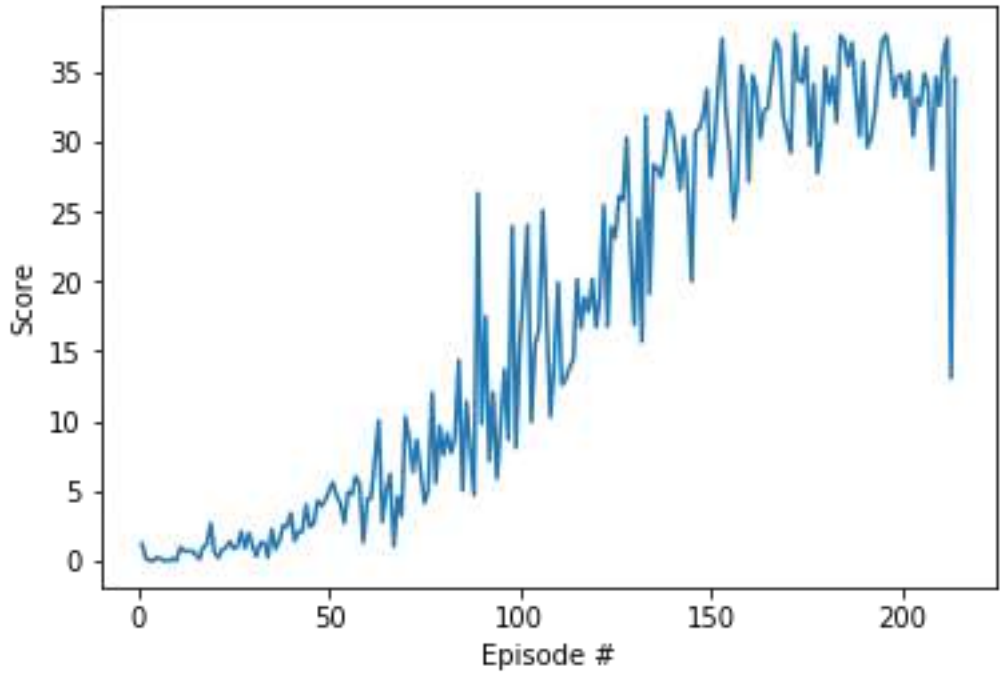
I went back to the project space in Udacity and studied again the benchmark implementation, and realised that their final working algorithm actually run updates only every twenty time steps, and that they used gradient clipping when training the critic network. I therefore decided to adopt both suggestions, and obtained slightly better results than the above.

At this point I was using a batch size of 128 to run gradient descent. I thought what would happen if I throwed more samples at the networks and noticed that by increasing the sample size I had better results than before.

I ended up using a batch size of 1024 and managed to resolve the environment after 425 episodes with an average score per hundred episodes of +30.06, as shown in the plot below.



I finally wondered whether going back to updating every four frames would still allow me to resolve the environment and discovered that it was the case! In this test the environment was resolved even faster in 214 episodes with an average score per hundred episodes of 30.08, despite the drop and recovery visible in the plot by the end of training.



Ideas for Future Work

1. Implement Prioritised Experience Replay in the Replay Buffer.

2. Implement distributional DDPG
3. Use the DDPG algorithm to solve an environment where we have images as inputs, i.e., learn from pixels.
4. Implement A2C, which is a different type of Actor Critic algorithms.