Project # 1 Navigation

The Environment:

This project uses DQN to train an agent to navigate and collect bananas in a large environment. The agent gets a reward +1 when collects yellow bananas and -1 when collects blue ones. This is an episodic task with a discrete space of 4 actions and a space state containing 37 dimensions.

Goal:

The goal is to train an agent to obtain an average score of +13 over 100 consecutive episodes.

Algorithm:

DQN was one of the first algorithms that reach state-of-the-art performance on the Atari benchmark. The paper was first published in 2013 by Volodymyr in the paper "Playing Atari with Deep Reinforcement Learning". The algorithm is based on 2 instances of the same network, the online network, is trained with offline samples at every step, the second instance, the target network, is used to reduce biases caused by changing the value function. That is because at every step the network approximates the value function which means that the target has been updated and the old one is not valid anymore. Using a delayed version of the online network (aka the target network) helps to stabilize the value function optimization process by making the process more stationary. Using a target network also reduces the chance of divergence, when the network is no longer converging to a minimum but in fact, doing the opposite. Another important feature of DQN is the use of experience replay. Experience replay is used to eliminate correlation between samples, stabilized the updates to the neural network (less noisy updates). Experience replay is a collection of "memories" which consist of tuples "state, action, reward, next state, done" that are stored and later used for learning. The selection of this memories is done at random thus decorrelating sequences of actions and thus eliminating potential biases.

Architecture:

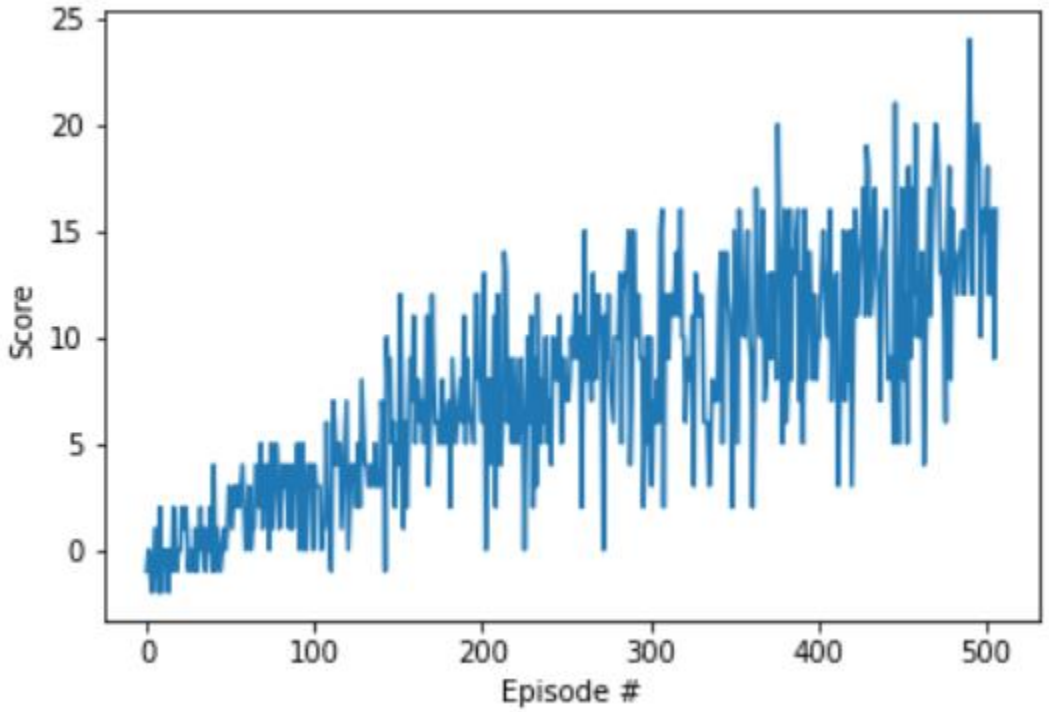
The Agent: It interacts with the environment, takes actions and collects memories. It is based on 2 neural networks. The online network and the target network only differ in the frequency of the updates. The online network is updated at every step while the target is updated every 4 steps. The process of updating the target network is referred as learning, here, the target network is updated based on $Qtarget = R + \ \gamma * \max(\text{Qtarget\_next})$. Also, at each step, the tuple of "state, action, reward, next state, done" is saved into a buffer from which samples are taken at random during the target network update.

The Neural Networks: 2 networks, target and local with same architecture. Each with 3 fully connected layers followed by a ReLU activation function. Hidden layer units 64, 64, 4 (action size) respectively.

Hyperparameters:

Buffer size = 1.e5 (replay buffer)
Batch size = 64
Gamma = 0.99 (discount factor)
TAU = 1e-3 (soft update target factor)
LR = 5e-4 (learning rate)
Target Network Update = 4 (frequency)

Results:



Future Ideas to Improve Algorithms Performance:

DQN can be sometimes unstable, a way to stabilize learning is to use a Double DQN algorithm which is a trick in the way the Q function is maximize that mitigates error propagation through the network. Other way DQN can be improved is using prioritized experience replay. Currently the learning step samples uniformly from the memory samples thus, all memories have the

same weight, in prioritized replay DQN, more weight is given to samples that incurred in larger error, following the intuition that overall performance is improved by focusing more on the experiences that have cause the lowest rewards. A more involving approach to improve DQN performance is to include all the independent improvements over the years (including DDQN and prioritized DQN). Google DeepMind researchers published a paper in Rainbow: Combining Improvements in Deep Reinforcement Learning that incorporates several improvements including:

1. DDQN
2. Prioritized DQN
3. Dueling DDQN
4. A3C
5. Distributional DQN
6. Noisy DQN

The combination of these improvements all together is called Rainbow algorithm, and it is currently the state-of-the-art algorithm in ATARI games.