Report

Learning Algorithm Description:

I opted Deep Q Network (DQN) to solve the given Banana environment. The implementation consists of two neural networks (NN) (designed with same network architecture), one for Q updates and other for Q targets using the Bellman equation. The NN architecture has 3 hidden layers besides input and output layer. The first hidden layer is a linear layer with dimensions (37, 256). The second hidden layer is a fully connected layer with dimensions (256,256). The third hidden layer is again a linear layer with reduced dimensions (256,128) which is then finally fed to an output layer of dimensions (128,4).

The agent Q networks namely Qlocal and Qtarget are initialized first with same uniform random weights. The agent then interacts with the banana environment, obtaining the tuples (state, action, reward, next state). Experience Replay is then used over these tuples, by storing them as samples into a finite buffer. A batch of samples from this buffer are randomly sampled each time and passed into the Qlocal and Qtarget networks. The Qvalues obtained from the networks for the batch are used inside the Bellman equation and mean squared loss is computed between them. The Qlocal network weights are updated using the Adam optimizer with a learning rate (5e-4) by minimizing the loss with respect to model parameters. Qtarget network parameters on the other hand are updated after every UPDATE_EVERY iterations using the equation below:

$$\theta\_target = \tau * \theta\_local + (1 - \tau) * \theta\_target$$

Here, τ is a update hyper-parameter (tuned to 1e-2) in my alogorithm.

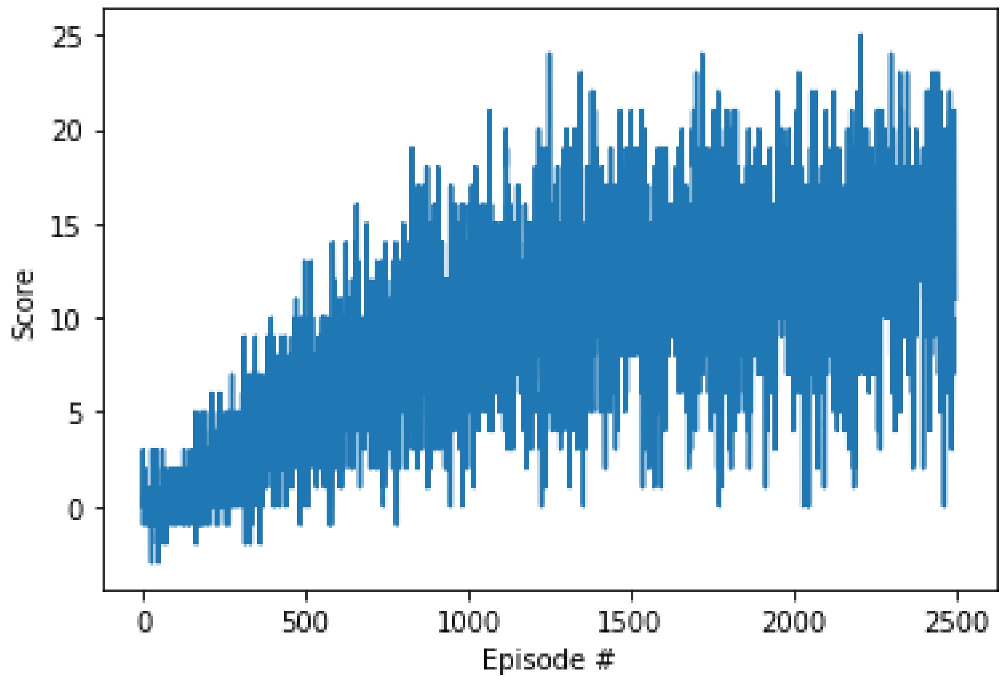
In brief, the DQN algorithm can be stated as follows:

1. Initialize $Q(s,a)$ with some initial approximation
2. By interacting with the environment, obtain the $tuple(s,a,r,s')$
3. Calculate Loss: ($L = (Q(s,a) - r)^2$) if episode has ended or
   $$L = (Q(s,a) - (r + \gamma\, max_{a' \epsilon A} Q(s',a')))$$
4. Update $Q(s,a)$ using Adam Optimizer with learning rate, by minimizing the loss with respect to model parameters
5. Repeat from step 2 until converged

Hyper-parameters Selection:

To the architecture designed in model.py, soft updates of Q-target network with tau=0.01 and UPDATE_EVERY = 20 iterations. Epsilon-greedy policy is chosen here for the exploration-exploitation partyes of deepRL. Mean Square Loss for a batch size of 64 is used as the loss function for current Q-local network values and expected Q-target network values. Adam Optimizer is then used at every step with learning rate (LR=5e-4) after performing back propagation at every step. The discount factor for episodic rewards (gamma) is set to 0.99. Number of training episodes considered are 2500, with eps_start = 1.0, eps_end = 0.01 and eps_decay = 0.9980 After performing hyper-parameter tuning and fixing the hyper-parameters to above mentioned values, the agent is able to solve the environment with an average score of 13.64 over 100 consecutive episodes. The weights of the learnt model are saved under the filename "model.pt" in the same folder.

Reward Plot:



Future Ideas:

Based on the designed NN architecture and hyper-parameter tuning, I could solve the environment with an average reward of 13.67 over 100 episodes. However, I find the NN architecture complex for the given problem. Trying out with lesser hidden units and less hidden layers if possible, is the future improvement I want to consider for my current solution.