Deep Reinforcement Learning Nanodegree

Project 1: Navigation

Report

The game is episodic with state space of 37 and discrete action space of size 4. Therefore an nn agent model following the DQN structure was selected to build the agent strategy.

A 100 thousand length buffer was defined to collect the agent experience while learning. This buffer is then periodically sampled to get a 64 long interactions batch which is then used to update the agent nn model through learning. The sampling takes place every 4 interactions, and thus learning occurs every 4 interactions.

The agent nn structure is a simple 3 fully connected (linear+relu) layers in addition to the output layer. The first layer has a size equal to the state size (37), the second and third layers has a size of 64, while the output layer has a size equal to the action space size (4).

The agent has the following capabilities:

1. Act: given the current state of the environment, it runs the state through the available agent nn model and selects the action according to the epsilon-greedy algorithm.
2. Step: given a complete experience, it saves it in the replay memory and if its the fifth experience since the last learning it calls the learning routine.
3. Learn: given the a sampled batch of size 64 experiences and the learning discount factor, it computes the error of the current agent model (which is computed as the mmean_square_error between the agent current q-values of the given states to the target q-values of the same states computed from the target nn) and propagates it back to update the agent model.
4. Soft_update: given the target-nn model update factor and the current agent-nn model, it updates the target-nn model in a way to ensure smooth slower change in the target-nn model so as to improve the whole DQN stability.

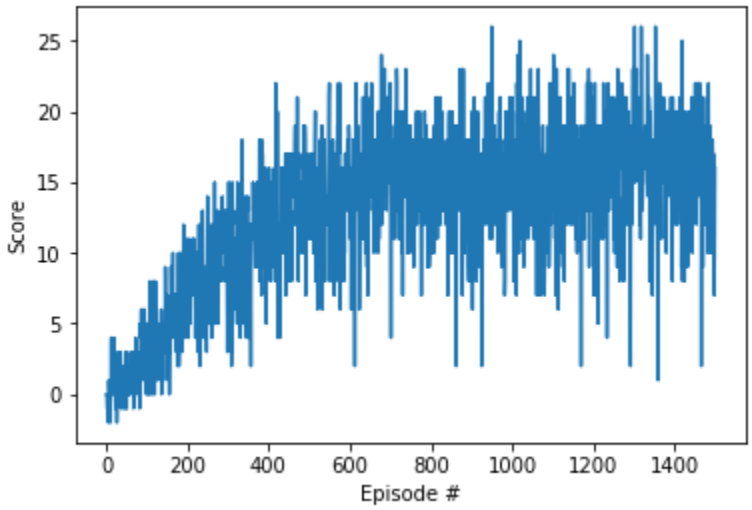
Learning:

For a predefined number of episodes (1500), the agent resets the environment, selects the action based on its current model and saves the experience into the replay buffer and accumulates the episode score.

While saving the experience in the replay buffer, the agent checks if it's the fifth experience since the last agent-model update. If so the agent samples a random batch and uses it to update its model.

Finally the agent learnt model is saved periodically every 100 episodes.

Below is the plot showing the learning performance of the agent. The average score exceeds +13 after 500 episodes.



Play using the learnt agent:

The saved agent model is loaded and used to play a single episode to test its performance. Note that the score might be below the benchmark of +13 since the benchmark is for the average score in a batch of 100 episodes and the learnt model is a stochastic one, hence the probability of a score less than +13 is not zero.

Future Ideas for improving theAgent performance:

The solution here implemented is a DQN structure. Indeed other structures can be used, however with this agent we were able to solve the environment with few training episodes (500). Hence for improvements I would work within the same DQN structure.

One of the things that might have a relatively strong impact on improving the performance is the soft update factor which defines the factor by which we update the target network from the local one. This can be reduced to better chase the optimal solution, however we might run into the limits where the system is no longer stable, so a bit of trials to look for a better soft update factor might actually yield some gains.

Another opportunity to explore is altering the nn model learning rate itself. This might again improve the learning by moving faster towards the optimal.