To solve the banana collector environment, I used the Deep Q-Network approach.

Background

Deep Q-Network has an network, which given an action evaluates how good the possible outcomes are. It also has backups of network that is updated at a slower frequency. The propose of these backup network is to make learning more stable, because its values change less and therefore allow for a more precise comparison.

The DQN update function is:

$$\Delta w = \alpha \cdot \overbrace{(\underbrace{R + \gamma \max_a \hat{q}(S', a, w^-)}_{\text{TD target}} - \underbrace{\hat{q}(S, A, w)}_{\text{old value}})}^{\text{TD error}} \nabla_w \hat{q}(S, A, w)$$

Were w represents the weights of the local network and w^- represents the value of the backup network. So as you can see by using the backup network we stabilize the second part of the td error and only compare the reward with the value on the local network.

One problem with DQN is that on the beginning of the learning it may not have enough information to figure out what the best action is. It depends a lot on what states and actions were explored. And therefore we might be propagating errors.

One way of addressing this problem is by using Double DQN. DDQN, on the second part of the td error, uses the local network to choose the max action and the backup network to evaluate this action.

$$R + \gamma q(S', argmax_a\, q(S', a, w), w^-)$$

If the local network chooses an action that is not the best in the backup network then the return is not that high. In the long run, it presents the algorithm to propagate incidental high rewards that may have been obtained by chance and don't reflect long-term returns.

Experiment

The following were the various applied variations:

- DQN with the frequency update of the local network update of every 4 iterations, frequency update of the target network of 1000 iterations and a exploration decay rate of 0.995 at every iteration (baseline model);
- DQN with the frequency update of the local network update of every 4 iterations, frequency update of the target network of 1000 iterations and a exploration decay rate of 0.95 at every iteration;
- DQN with the frequency update of the local and a network update of every 4 iterations, frequency soft update of the target network of 1000 iterations and a exploration decay rate of 0.995 at every iteration. Soft update means only part of the network changes. Here only 1e-3% of the network changed;

- DQN with the frequency update of the local and a network update of every 4 iterations, frequency soft update of the target network of 1000 iterations and a exploration decay rate of 0.995 at every iteration. Here only 1e-2% of the network changed;
- DDQN with the frequency update of the local network update of every 4 iterations, frequency update of the target network of 1000 iterations and a exploration decay rate of 0.995 at every iteration;
- DDQN, with the frequency update of the local network update of every 4 iterations, frequency soft update of the target network of 4 iterations and a exploration decay rate of 0.995 at every iteration. Here only 1e-3% of the network changed;

Figure 1 shows the rewards per episode for each of these models. By order of presented models the colors of the plot are: blue ,orange,green, purple, red, and brown. As we can see the DQN model with decay exploration rate of 0.95 outperforms the other models.

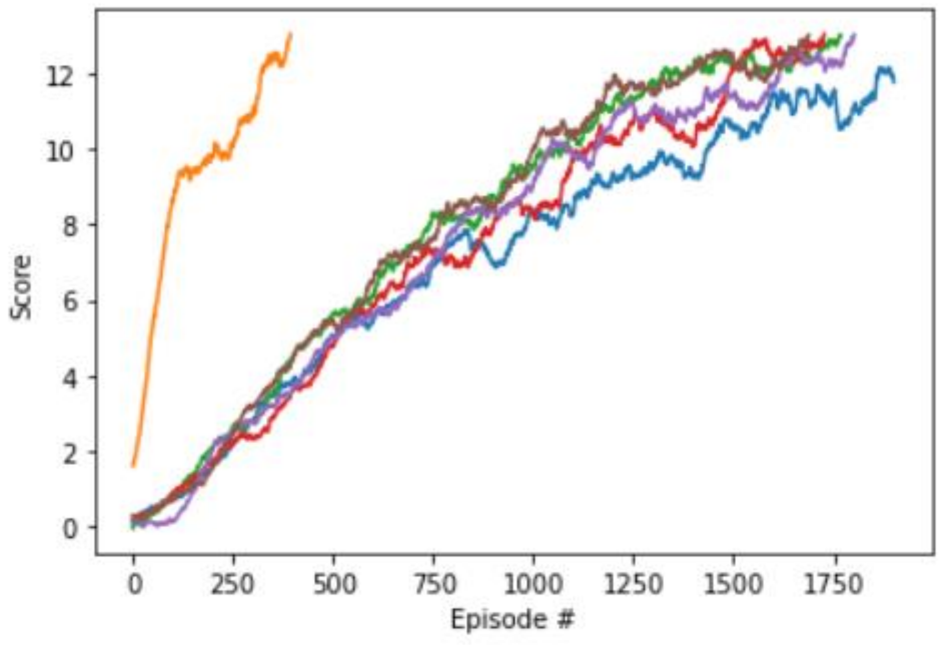

Figure 1 Score analysis per episode of the different models.

Conclusion

The model that achieved the best results by far was the DQN with an exploration decay rate of 0.95. Followed by the network that used DDQN and a 1e-3% soft update.

I also observed that the soft update with a 1e-3% change was best than the 1e-2% change. The latest one may change too quickly to make the network stable.

Future work

In the future I would like to test the impact on performance of prioritized replay, with the default hyperparameters of the baseline model. I expect this approach to have a positive impact on performance.