Report Navigation

Author: Dorzhi M.

Approach description

We apply Deep Q-Learning to solve banana collection problem. The approach we apply is based on '[Human-level control through deep reinforcement learning]' (Mnih et al.).

Banana collection problem is a typical reinforcement learning problem. We have an agent and an environment which responds to the agent's actions, and the agent receives rewards after each action. Reinforcement learning problems can be addressed using Q-learning, where we assign a value *Q(s,a)* to each action *a* given the state of the environment *s* which represents a long-term expected reward creating a Q-table. This Q-table can be used as a lookup table with the optimal agent's actions for each state.

Q-table initiated to zeroes. The values *Q(s,a)* are updated iteratively at every step of the agent's interaction with the environment using **Temporal Difference** equation $Q(s,a) = Q(s,a) + \alpha(r(s) + \gamma Q(s',a') - Q(s,a))$, where *r(s)* is the reward for the state *s, s'* is the resulting state after taking action *a* in state *s.*

The long-term values for the Q-table after the training are represented by the **Bellman equation** $Q(s,a) = r(s) + \gamma max(Q(s',a'))$. In the long run we want the Q-table to satisfy the Bellman equation.

The environments can be quite complex, and the direct usage of a Q-tables can be challenging from the computational perspective. We can use neural networks to approximate the Q-table. However, the naive application of neural net does not provide drastic improvement for the agent's performance. In the above mentioned paper the authors suggested using several techniques that dramatically improve the performance of the agent.

These techniques are **experience replay** and **fixed Q-targets**.

The sequence of the agent's experience tuples can be correlated, and this correlation can sway the results of the learning process. By instead keeping track of a **replay buffer** and using **experience replay** to sample from the buffer at random, we can prevent action values from oscillating or diverging catastrophically.

In Q-Learning, we update a guess with a guess, and this can potentially lead to harmful correlations. To avoid this DQN uses a separate target network which weights are not changed during the learning step (fixed Q-targets).

The environment's state space has dimension of 37, actions space is discrete and has dimension of 4.

We used a fully connected neural network with 2 hidden layers. The following neural net is used for parametrizing Q function:

- 37-dimension input layer
- 2 hidden layers with dimension of 64
- 4-dimension output layer

We set following hyperparameters:

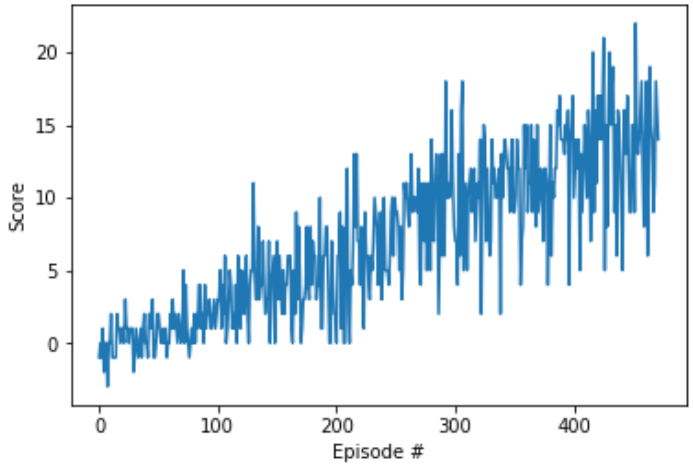
- maximum number of training episodes (by default 2000)
- maximum number of timesteps per episode (by default 1000)
- starting value of epsilon, for epsilon-greedy action selection (by default 1.0)
- minimum value of epsilon (by default 0.01)
- multiplicative factor (per episode) for decreasing epsilon (by default 0.995)

The maximum number of training episodes was taken from the example project, the maximum and minimum values as well as multiplicative factor are as usual used in this course. For the maximum number of timesteps per episode, I first tried to set it to a lower number (about 100) in order to accelerate the learning process, however the performance in this case was substantially worse.

In order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

Agent Performance

We were able to solve the problem in 372 episodes.



Ideas for future

As next steps, we can improve the performance of the agent by using prioritized experience replay, as some of the transitions are more important for the agent performance (those with higher error), and we can sample those transitions with higher probability.