Navigation Project Report

05 July 2019 11:05

For this project, the scope is to train an agent to navigate and collect bananas in a large square world.

A reward of +1 is provided for collecting a yellow banana, and a reward of -1 is provided for collecting a blue banana. Thus, the goal of the agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions and contains the agent's velocity, along with ray-based perception of objects around the agent's forward direction. Four discrete actions are available: move forward, move backward, turn left and turn right. Therefore the agent has to learn how to best select actions.

The task is episodic, and in order to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

After conducting a brief research, I decided to use a slightly modified version of the Deep Q-Learning algorithm (implemented by Google Deepmind), which represents the optimal action-value function $q*$ as a neural network, instead of a table. In order to stabilize the learning, this algorithm uses two key features: Experience Replay and Fixed Q-Targets.

When the agent interacts with the environment, the sequence of experience tuples can be highly correlated. By learning
from each of these experience tuples in sequential order runs the risk of getting swayed by the effects of this correlation. 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.

The replay buffer contains a collection of experience tuples (S - state , A - action , R - reward, S' - estimated state). The tuples are gradually added to the buffer as we are interacting with the environment.

The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

In Q-Learning, we update a guess with a guess, and this can potentially lead to harmful correlations. To avoid this, we can update the parameters w in the network q^ to better approximate the action value corresponding to state *S* and action *A* with the following update rule:

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

where $w-$ are the weights of a separate target network that are not changed during the learning step, and (*S*, *A*, *R*, *S*') is an experience tuple.

Network architecture & Hypermeters

4 fully connected layers with Relu activation:

The 1st FC layer maps the states (37) to 256 rectifier units
The 2nd FC layer connects the 256 units to 128 rectifier units
The 3rd FC layer connects the 128 units to 64 rectifier units
The 4th FC layer maps the 64 units to actions (4)

I chose the following hyperparameters:

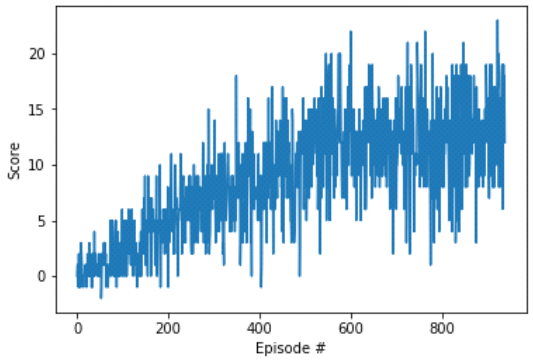
BUFFER_SIZE = 1e5 (number of experience tuples stored in the replay buffer)
BATCH_SIZE = 64 (sampling a small batch of tuples from the replay buffer in order to learn)
GAMMA = 0.99 (decay rate for the expected reward)
TAU = 1e-2 (used for a soft update of the target parameters)
LR = 5e-4 (learning rate of the network)
UPDATE_EVERY = 4 (how often to update the network)

Please find below a plot of rewards per episode to illustrate that the agent is able to receive an average reward (over 100 episodes) of at least +13 in 839 episodes.

```
Episode 100     Average Score: 1.07
Episode 200     Average Score: 3.29
Episode 300     Average Score: 5.93
Episode 400     Average Score: 7.69
Episode 500     Average Score: 9.05
Episode 600     Average Score: 12.18
Episode 700     Average Score: 11.89
Episode 800     Average Score: 11.69
Episode 900     Average Score: 12.12
Episode 939     Average Score: 13.07
Environment solved in 839 episodes!     Average Score: 13.07

<matplotlib.figure.Figure at 0x7f41976e8a20>
```

In [9]:
```
# plot the scores
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(len(scores)), scores)
plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Future Work

In order to improve the score the following changes will be considered for future work:

1. A Double DQN in order to address the overestimation of the Q - values. Within the update rule for Q -learning, the TD target is calculated by picking up the action with the max Q value, which is prone to error during the initial stages when the Q - value is still evolving and doesn't have enough information to figure out the best action.
   Double Q - Learning selects the best action using one set of parameters but evaluates the action using a different set of parameters, in practice the parameters of the fixed Q-targets.

2. Prioritized Experience Replay : By sampling the batches uniformly we might miss out rare significant occurrences.
   The criteria of assigning priorities to the experience tuples is using the TD error delta, the bigger the error, the more we expect to learn from it. The priority is calculated by taking the magnitude of the TD error delta, including a hypermeter e (to control the selection of the tuples where the TD error is very close to 0, which might still be useful) and then sampling a probability distribution. This can still lead to greedily using a small set of samples replayed over and over resulting in overfitting, but this can be fixed by reintroducing a small factor of random sampling using the formula:

   $$P(i) = \frac{{p_i}^a}{\sum_k {p_k}^a}$$

   Furthermore the Update Rule has to be updated in order cope with the non-uniform sampling by introducing an importance sampling weight (otherwise the Q-values learned will be biased according to these priority values).

3. Dueling DQN: Use two streams, one that estimates the state value function and another one to estimate the advantage values for each action, while the desired Q-values are obtained by combining the two of them. The reasoning behind is that the value of most states don't vary a lot across action so we can directly estimate them.