UDACITY Deep Reinforcement Learning
Sang-Ho Kim
Project #1 Navigation
3 June 2019

Navigation
Deep Q Network

The environment of Unity Banana was solved using Deep Q Network. The score over 13.0 was obtained at 434 episodes with epsilon greedy action selection, replay buffer (size of 1e5) and soft update (tau of 1e-3). The learning rate was 0.0025. The network was updated every 4 time steps with maximum 100 steps for each episode. The final results met all the requirements of rubric.

LEARNING ALGORITHM

Deep Q Network or DQN outperforms Q-learning with regards to estimating value for unseen states by replacing two dimensional array of (state and action) with neural network. The state is the input and the output is the Q-value for each action. To train the network the target value for Q-learning is updated through optimization of the loss function function shown below.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

The first two terms in the parentheses are the target Q-value and the third term is the the Q-value output from the network.

With DQN, two additional techniques are essential. They are Experience Replay and Separate Target Network. With Experience Replay, the highly correlated training samples are randomly selected and stored to update knowledge. Separate Target Network scheme helps

avoiding swinging back and forth to obtain more stable trainings. The target Q Network is reset separately from the local Q Network.

The pseudo code for the DQN algorithm is as follows.

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

REPLAY BUFFER

Many replay buffer sizes were tried before obtaining the final size of 1e5. The smaller the buffer size resulted in no learning at all (the score oscillated around zero.) Moreover, the magnitude of the oscillation reduced at the further time steps. This implies that the learning was stuck at a local minimum at the very early stage and the solution was biased. To avoid this bias, the replay buffer size was increased.

LEARNING RATE

The results were very sensitive to the learning rate. The smaller learning rate led to a small increase in the score value with further time steps. The learning rate was found by increasing little by little before the solution showed no learning due to divergence.

SOFT UPDATE

The network was updated every 4 steps with the update rate of 1e-3. This update rate is also closely related to replay buffer as the actions were sampled from the replay buffer during the update. When the soft update value was relatively large, the network seemed to be lost and biased.

MAXIMUM ITERATION NUMBER PER EPISODE

The maximum iteration number for each episode was a very important parameter. When it was small, the network did not learn much and the learning progress was very slow. The large iteration seemed to make the network learn more about the environment during more exploration.

SCORE

The history of scores are plotted in the following figure. Although the average score steadily increased, the magnitude of the fluctuation also increased. This may imply that the network seems more efficiently learn from the increased liberty of choosing action by epsilon greedy selection. This algorithm seems very effective in finding solutions in more complicated environment.

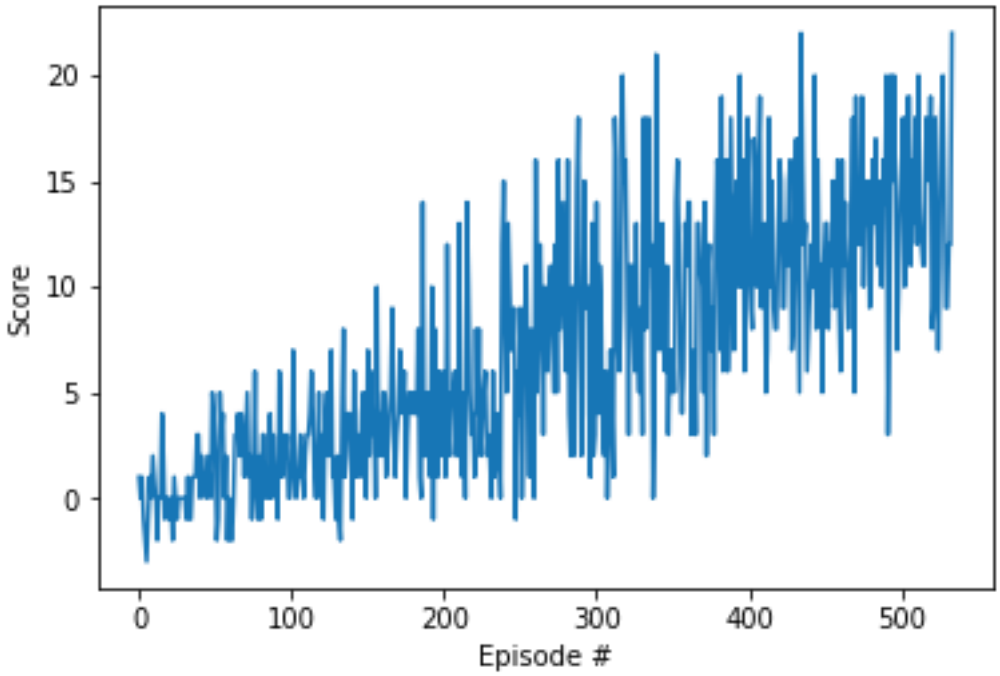

Figure: History of Scores

IDEAS FOR FUTURE WORK

At this moment, only DQN was explored. More advance techniques such as Double DQN, Dueling DQN or Rainbow DQN may be used.