Deep Reinforcement Learning Nanodegree

Deep-Q Learning in Action: Navigation

Ayham Zaza

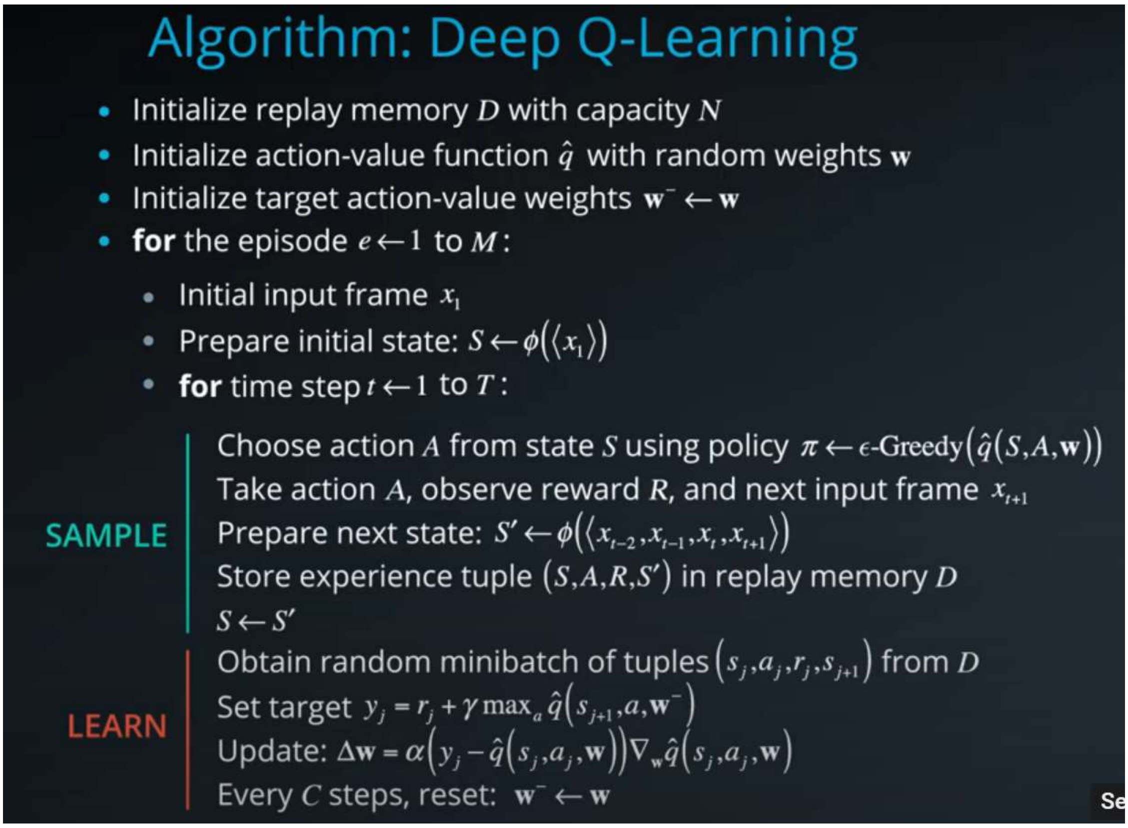
Contents

Introduction

- An Agent **G** located at **State S**, interacts with its surrounding *environment* by trying a **Action A** to transfer to a **New State S'**. A **Reward R** is associated with each action; and the goal of the agent is to navigate the environment until it reaches an *Objective* **O** while achieving a maximum possible reward. At a given state, the agent *experience* is summarized by the tuple (**S,A,R,S'**).
- When interacting with the environment, the sequence of experience tuple taken by an agent are corelated; current action impacts future actions. As a result, the agent may stuck in a given experience that might not be the optimal.
- A first remedy is through "**Experience Replay**" strategy where a memory is utilized to store experiences. Throughout the learning process, the agent, choses a random stored experience and quantifies/reinforces how much it learns.
- To reach the goal, the agent makes a guess on its next action and evaluates it. Once in a new state, it makes another guess to move to another state and evaluates it. In essence, the guess made by the agent is updated by another guess, that in turns is updated with another guess and so on. To minimize the error in such a process, a strategy termed as "**Fixed-Q Targets**" was suggested. It offers a kind of temporal decoupling of target optimization objective where **two step predictions** are made according to the following equation (by keeping $\boldsymbol{w}^{-}$ fixed during the learning for a given experience tuple):

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

Deep-Q Learning Algorithm



Model Description

Network Architecture

- Feedforward network implemented in PyTorch Linear module.

```
self.fc1 = nn.Linear(state_size, fc1_units)
self.fc2 = nn.Linear(fc1_units, fc2_units)
self.fc3 = nn.Linear(fc2_units, action_size)
```

- Input: 37 nodes – corresponding to state size
- Output: 4 nodes – corresponding to action size
- To enable learning complex non-linear functions, the two hidden layers (fc1, fc2) are followed by ***relu*** activation function.
- (fc1,fc2) were both chosen to be of size 64.

Hyperparameters

```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 64         # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR = 5e-4               # learning rate
UPDATE_EVERY = 4        # how often to update the network
```

Results

The decision for the number of layers/hidden units, as well the choice of DQN network solved the environment by achieving the target score in less than 600 episodes. Other design choices or enhancement for DQN can also be utilized when needed. See the figure_1

```
Episode 100	Average Score: 1.09
Episode 200	Average Score: 4.46
Episode 300	Average Score: 8.25
Episode 400	Average Score: 10.91
Episode 500	Average Score: 12.89
Episode 518	Average Score: 13.02
Environment solved in 518 episodes!	Average Score: 13.02
```

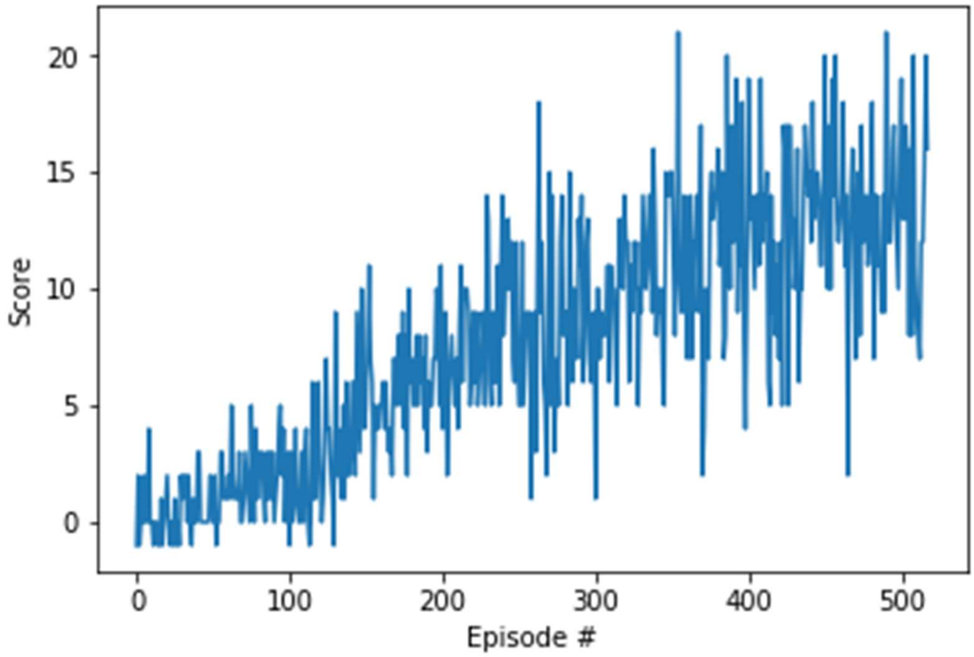


Figure 1: A plot of rewards per episode: the agent is able to receive an average reward (over 100 episodes) of at +13

Future Improvements

- Trying different network architectures
- Solving the environment with less episodes
- Solving the environment with higher average reward
- Although not needed, but checking how various improvements to DQN would perform is interesting -Prioritized experience replay, Dueling DQN Learning from multi-step bootstrap targets, A3C, Distributional DQN, Noisy DQN-

References

- Udacity Deep Reinforcement learning Nanodegree Program. Lecture Notes and Videos. (https://www.udacity.com/course/deep-reinforcement-learning-nanodegree--nd893)
- https://github.com/udacity/deep-reinforcement-learning
- Deep Reinforcement Learning with Double Q-learning. Hado van Hasselt, Arthur Guez, David Silver. 2015
- Human-level control through deep reinforcement learning. Volodymyr Mnih, et al. 2015
- Reinforcement Learning: An Introduction. Book by Richard S. Sutton and Andrew Barto. Second Edition. 2018