

# Deep Reinforcement Learning Nanodegree

## Deep-Q Learning in Action: Navigation

### Contents

<b>Introduction .....</b>	<b>2</b>
<b>Deep-Q Learning Algorithm.....</b>	<b>3</b>
<b>Model Description .....</b>	<b>3</b>
<b>Network Architecture .....</b>	<b>3</b>
<b>Hyperparameters .....</b>	<b>4</b>
<b>Results .....</b>	<b>4</b>
<b>References.....</b>	<b>5</b>

## 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 correlated; 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  $w^-$  fixed during the learning for a given experience tuple):

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

TD targetold value

## Deep-Q Learning Algorithm

### Algorithm: Deep Q-Learning

- Initialize replay memory  $D$  with capacity  $N$
- Initialize action-value function  $\hat{q}$  with random weights  $\mathbf{w}$
- Initialize target action-value weights  $\mathbf{w}^- \leftarrow \mathbf{w}$
- **for** the episode  $e \leftarrow 1$  to  $M$ :
  - Initial input frame  $x_1$
  - Prepare initial state:  $S \leftarrow \phi(\langle x_1 \rangle)$
  - **for** time step  $t \leftarrow 1$  to  $T$ :

#### SAMPLE

Choose action  $A$  from state  $S$  using policy  $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}(S, A, \mathbf{w}))$   
Take action  $A$ , observe reward  $R$ , and next input frame  $x_{t+1}$   
Prepare next state:  $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$   
Store experience tuple  $(S, A, R, S')$  in replay memory  $D$   
 $S \leftarrow S'$

#### LEARN

Obtain random minibatch of tuples  $(s_j, a_j, r_j, s_{j+1})$  from  $D$   
Set target  $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, \mathbf{w}^-)$   
Update:  $\Delta \mathbf{w} = \alpha (y_j - \hat{q}(s_j, a_j, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(s_j, a_j, \mathbf{w})$   
Every  $C$  steps, reset:  $\mathbf{w}^- \leftarrow \mathbf{w}$

Se

## 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 500 episodes. Other design choices or enhancement for DQN can also be utilized when needed. See the figure\_1

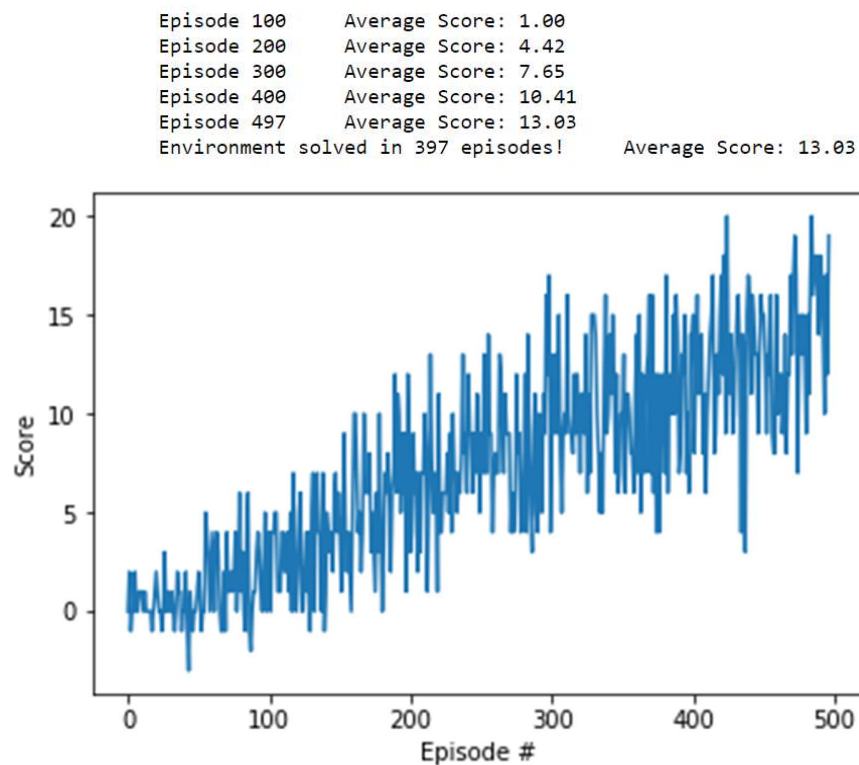


Figure 1: A plot of rewards per episode: the agent is able to receive an average reward (over 100 episodes) of at +13

## 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