

# Udacity Deep Reinforcement Learning Nanodegree

## Project #1 Navigation

### Report

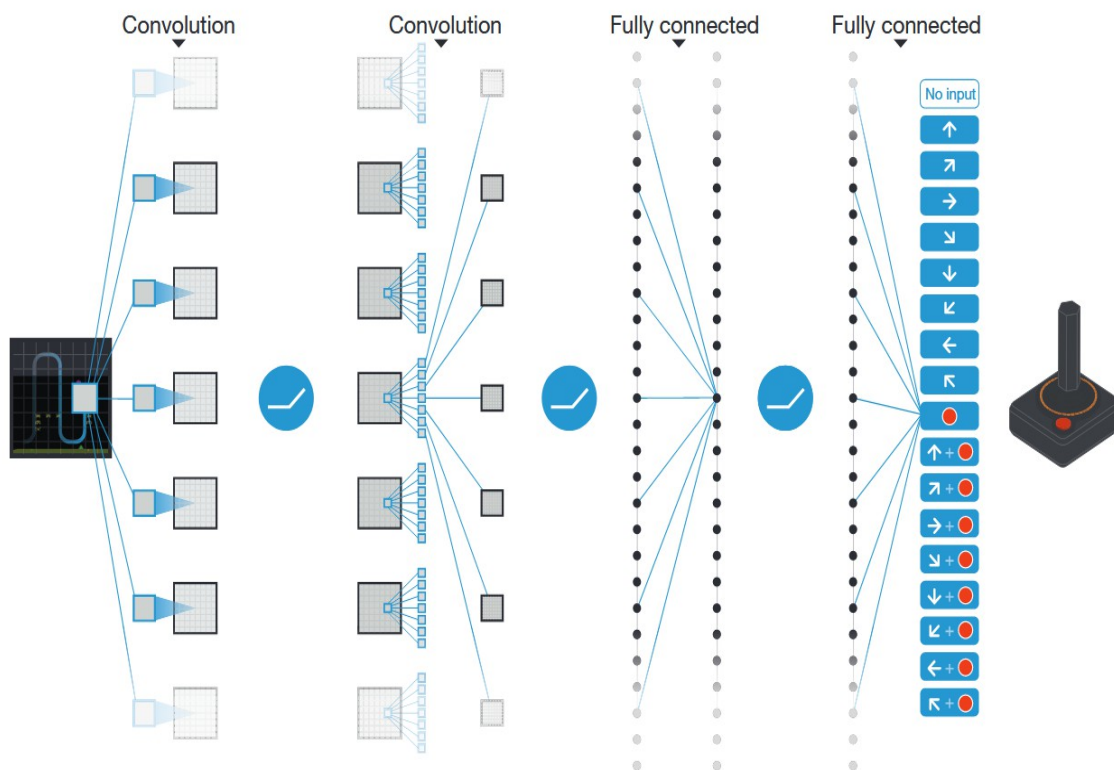
#### 1. Learning Algorithm

The algorithm implemented is a *Deep Q-Network algorithm*, that is, an algorithm which approximates the state-value Q function with the use of a Neural Network.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Source: Sutton and Barto, *Reinforcement Learning, 2nd Edition*

Such algorithm usually takes as input several frames of the game and outputs the state values for each action, and the network is composed of Convolutional and Fully Connected layers, as shown below.



Source: <https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>

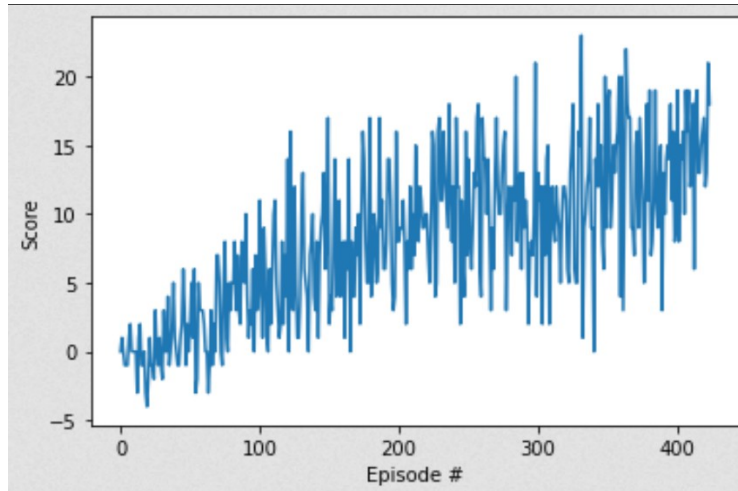
In this particular environment, however, the input state is a 37 dimensions vector containing information obtained from the agent observing the environment about the velocity, along with ray-based perception of objects around agent's forward direction. Given this input, the agent has to choose among 4 possible different discrete actions (move forward, move backward, turn left and turn right). Given the nature of the problem, a deep neural network have been created which is composed of an input of the vector state of 37 dimensions, followed by two Fully connected layers (of size 128 and 64), and a final layer mapping to the possible 4 actions. It has to be noted that this implementation does not use any Convolutional layer, this has been preferred as the provided network is able to solve the environment while keeping the implementation simple and straightforward. As activation function between the layers, the ReLu function has been chosen, which is well-known to be a valuable non-linear activation function used when training deep neural networks.

The algorithm uses the Experience Replay optimization, for storing the episode steps in memory for off-policy learning, where samples are drawn from the replay memory at random. Additionally, the Q-Network is optimized towards a frozen target network that is updated using a soft-update strategy with the use of an hyperparameter. The latter makes training more stable by preventing short-term oscillations from a moving target. The former tackles autocorrelation that would occur from on-line learning, and having a replay memory makes the problem more like a supervised learning problem.

For solving the environment, the hyperparameters have been chosen as following:

- **GAMMA** (discount factor): 0.99
- **TAU** (for soft-update of target parameters): 1e-3
- **LR** (learning rate): 5e-4
- **BUFFER\_SIZE** (replay buffer size): 1e5
- **BATCH\_SIZE** (minibatch size): 64
- **EPS\_DECAY\_RATE** (decay rate for epsilon value): 0.95 (ranging from 1.0 to 0.01)

## 2. Plot of Rewards



*Plot of rewards over episodes.*

The agent was able to solve the environment (i.e. get *an average reward of +13 over 100 consecutive episodes*) in **324** episodes.

## 3. Ideas for Future Work

There are many possible improvements for the DNQ algorithm:

1. Double DQN: Deep Q-Learning tends to overestimate action values. [Double Q-Learning](#) has been shown to work well in practice to help with this.
2. Prioritized Experience Replay: Deep Q-Learning samples experience transitions *uniformly* from a replay memory. [Prioritized experienced replay](#) is based on the idea that the agent can learn more effectively from some transitions than from others, and the more important transitions should be sampled with higher probability.
3. Dueling DQN: Currently, in order to determine which states are (or are not) valuable, we have to estimate the corresponding action values *for each action*. However, by replacing the traditional Deep Q-Network (DQN) architecture with a [dueling architecture](#), we can assess the value of each state, without having to learn the effect of each action.
4. [Rainbow](#) extensions.

Last, but not least, the use of pixels as input would be a nice to have feature, updating the Neural Network in order to accommodate some initial Convolutional Layers as needed.

#### 4. Sources

- <https://paperswithcode.com/method/dqn>
- <http://incompleteideas.net/book/RLbook2020.pdf>
- <https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf>
- [https://github.com/udacity/deep-reinforcement-learning/tree/master/p1\\_navigation](https://github.com/udacity/deep-reinforcement-learning/tree/master/p1_navigation)
- <https://deeptai.org/machine-learning-glossary-and-terms/relu>