PROJECT 1: NAVIGATION

P.M. Lucaci

DEEP REINFORCEMENT LEARNING NANODEGREE, Udacity

Contents

Learning Algorithm

DQN stands for "Deep Q-Network", which is an extension to the Reinforcement Learning ("RL") algorithm "Q-Learning".

Q-Learning uses tuples (S,A,R,S') (i.e. State, Action, Reward, Next State) to estimate the optimal (or nearly optimal) state-action value (also known as a Q-Function). In turn, the Q-Function maximises the agent's expected cumulative reward.

Therefore, we can say that DQN is using deep neural networks ("NN") to estimate the expected cumulative reward by computing the optimal action-value function:

$$Q_*(s,\ a) = \max_\pi \mathbb{E}[r_t + \gamma r_{t+1} + \gamma r_{t+2} + ... \mid s{=}s,\ \ a{=}a, \pi]$$

where Q_* is the maximum sum of expected rewards r_t, discounted at each time step t, by a factor γ, based on taking action a, given state observation s, and the policy π=P(a | s).

DQN algorithm brings 2 new features to Q-Learning:

- **Replay Memory**, which stores experience tuples $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t, in a data set $D_t = \{e_1, e_2, \ldots e_t\}$. Small batches of experience tuples *U(D)* are being accessed from the Replay Memory during learning, to train the NN "Q-Network-Local". For practical reasons, only the last ***N*** experience tuples are being stored. ***The motivation for a Replay Memory?*** It disrupts the correlation between consecutively sampled experience tuples.
- A secondary network ,"Q-Network-Target", acting as a **target network** computing the expected action-state values for the "Q-Network-Local" NN. Accordingly, at each iteration i, a loss function is used to determine how far the current Q-Function is from the target:

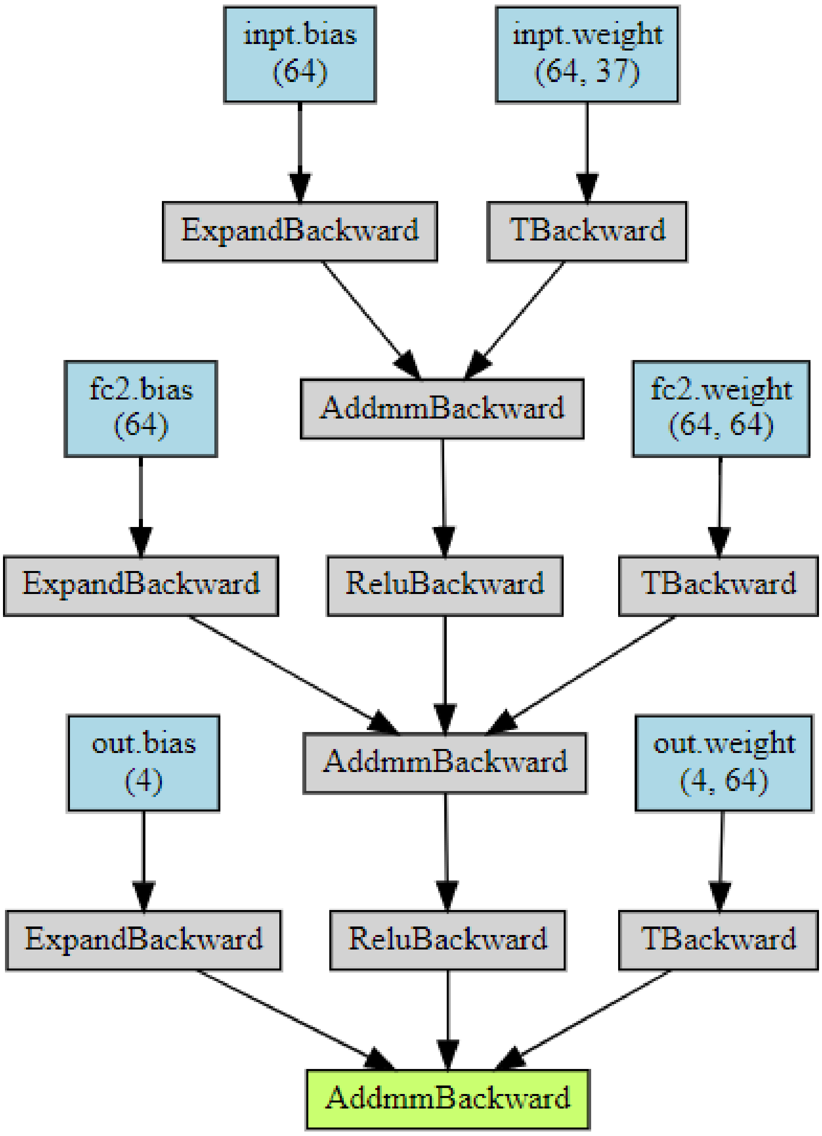
$$L_i\ (\theta_i) = \mathbb{E}_{(s,a,r,s')\sim(D)}[(\mathrm{r}t + \gamma \max_{a'} Q(s', a';\ \theta_i^-) - Q(s, a;\ \theta_i))^2]$$

The motivation for a secondary network? To avoid updating a guess (the weights of "Q-Network-Local" with another guess (a slight increase using a constant, like Q-Learning does).

Hyperparameters

```
n_episodes = 2000                  # no. of episodes to train
eps_start = 1.0                    # epsilon upper limit (before any decay)
eps_end = 0.01                     # epsilon lower limit (minimum value)
eps_decay = 0.995                  # epsilon decay rate
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 = 0.001                         # learning rate
UPDATE_EVERY = 4                   # how often to update the network
```

NN Model Architecture



LinearLayer(in = 37, out = 64) - > ReLu - > LinearLayer(out = 64) - > ReLu - > LinearLayer(out = 4)

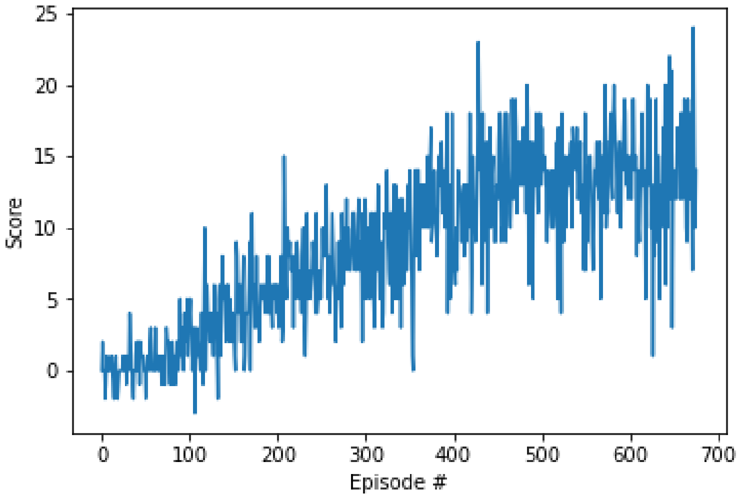
N.B. Same architecture has been used for both networks: QTarget and QLocal

Plot of Rewards

Episode 100	Average Score: 0.64
Episode 200	Average Score: 3.95
Episode 300	Average Score: 7.25
Episode 400	Average Score: 9.90
Episode 500	Average Score: 12.95
Episode 600	Average Score: 13.29
Episode 676	Average Score: 14.00

Environment solved in 576 episodes!

Average Score: 14.00



Ideas for Future Work

Improving the report

Including project details

GIF of Trained Agent

Action Space

State Space

Rewards

Solving the Environment

Documenting Future Improvements of Learning Algorithm

Implementing the Rainbow Learning Algorithm

Documenting the Rainbow Learning Algorithm