Project Report

SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence

probabilistic learning

Aakash Nandrajog
1734002

Prof: luca Iocchi

FrozenLake

The idea of Frozen-lake is that The agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water. Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.

1. Using Q-Learning Algorithm
2. Using Q-neural network.

FrozenLake using Q-learning

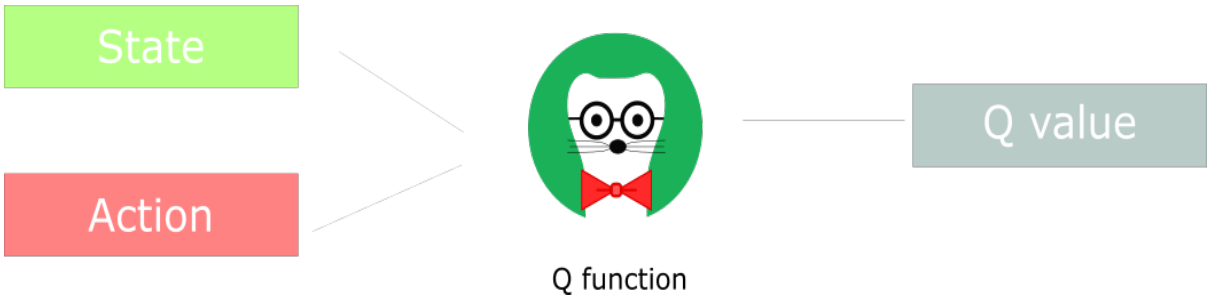
The goal of this game is **to go from the starting state (S) to the goal state (G)** by walking only on frozen tiles (F) and avoid holes (H).

```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

Q-Learning – Algorithm learning the Action Value Function

1) Q-learning takes two inputs: “State” & “Action”.

2) Returns the expected future reward of that action at that state.



The core of Q-Learning is to estimate a value for every possible pare of a state (s) and an action (a) by getting rewarded. Imagine the following graph, which consists of three states, while your agent is currently in _s0_. It can choose between two actions, one of which results in a good state _s1_ the other one results in a bad state _s2_. Accordingly, the transition leading to the good (bad) state gives a reward of 100 (-100). If the agent performs action _a0_, the q-value of _s0_ will probably become negative (Q(s0, a0) < 0)), while Q(s0, a1) > 0.

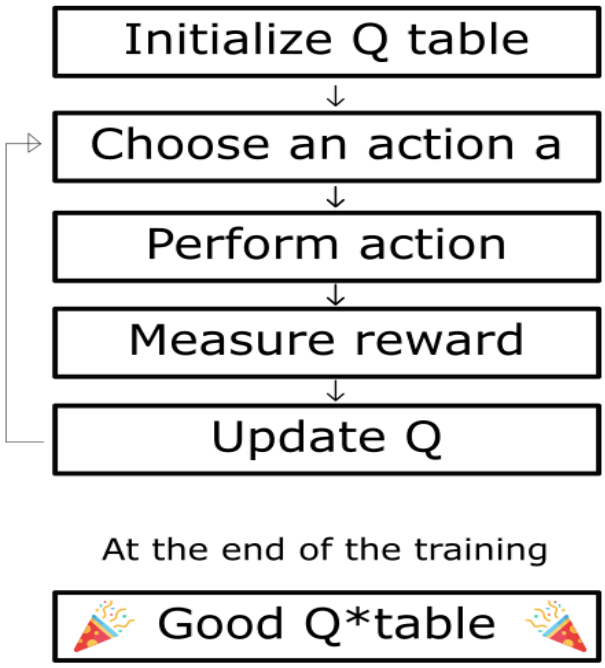
The update of the q-value is done according to the following equation.

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \alpha[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \gamma \underbrace{max\,Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

Learning Rate

Discount rate

The Q-learning algorithm Process



<u>High level Implementation</u>:

The Q learning algorithm's pseudo-code :

1. Initialize Q-values ($Q(s, a)$) arbitrarily for all state-action pairs.
2. For life or until learning is stopped...
3. Choose an action (a) in the current world state (s) based on current Q-value estimates ($Q(s, \cdot)$).
4. Take the action (a) and observe the the outcome state (s') and reward (r).
5. Update $Q(s, a) := Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$

Step 1: Initialize Q-values

- We build a Q-table, with m cols (m= number of actions), and n rows (n = number of states). We initialize the values at 0.

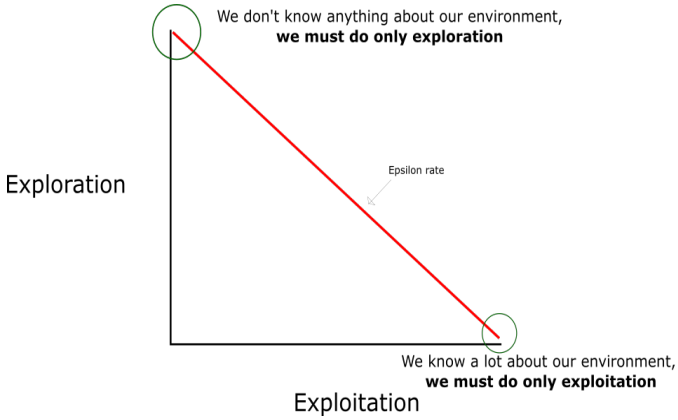
Step 2: Choose an action:

Choose an action a in the current state s based on the current Q-value estimates.

- what action can we take in the beginning, if every Q-value equals zero?
- we'll use the epsilon greedy strategy:

epsilon greedy strategy:

- We specify an exploration rate “epsilon,” which we set to 1 in the beginning. This is the rate of steps that we’ll do randomly.
- We generate a random number. If this number > epsilon, then we will do “exploitation”. Else, we’ll do exploration.
- The idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively as the agent becomes more confident at estimating Q-values.



- **Steps 3–4: Evaluate!**
  Take the action *a* and observe the outcome state *s’* and reward *r*. Now update the function Q(s,a).
  - We take the action *a* that we chose in step 3, and then performing this action returns us a new state *s’* and a reward *r*
  - Then, to update Q(s,a) we use **the Bellman equation:**

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

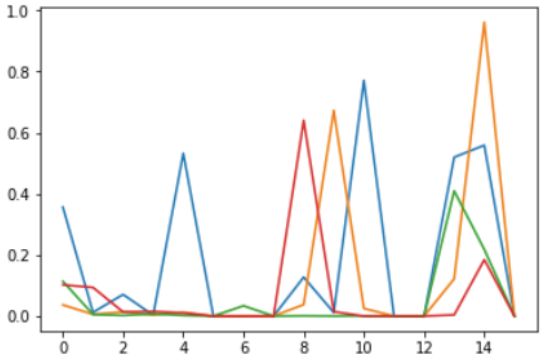
Maximum expected future reward **given the new s’ and all possible actions at that new state**

Output

Score over time = 6.6666666

Q table was initialize at 0 but after performing Q –learning algorithm the results are down below.

```
Score over time: 6.666666666666667e-05
[[3.26623933e-01 1.08836218e-01 2.20346744e-02 2.62108646e-02]
 [2.77294081e-03 5.07220414e-03 7.80388018e-03 2.89324738e-01]
 [4.62695749e-03 6.79099117e-03 4.39680102e-03 1.39298498e-01]
 [8.15083460e-03 4.91913331e-03 5.07007929e-03 2.41931470e-02]
 [2.60642484e-01 1.29953230e-01 4.18463302e-02 8.48845017e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [2.04098697e-02 5.82696928e-05 4.30634664e-05 3.06294958e-09]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.82220230e-02 8.83783975e-02 1.26328169e-02 2.37330405e-01]
 [1.92824333e-03 2.71108005e-01 8.60609020e-02 1.76069327e-02]
 [7.51434848e-01 1.05871750e-03 6.37003732e-03 3.02068376e-05]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [8.73852309e-02 1.21339144e-01 5.04579520e-01 7.68476198e-03]
 [2.05643162e-01 9.17989538e-01 1.03065476e-01 1.67924968e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

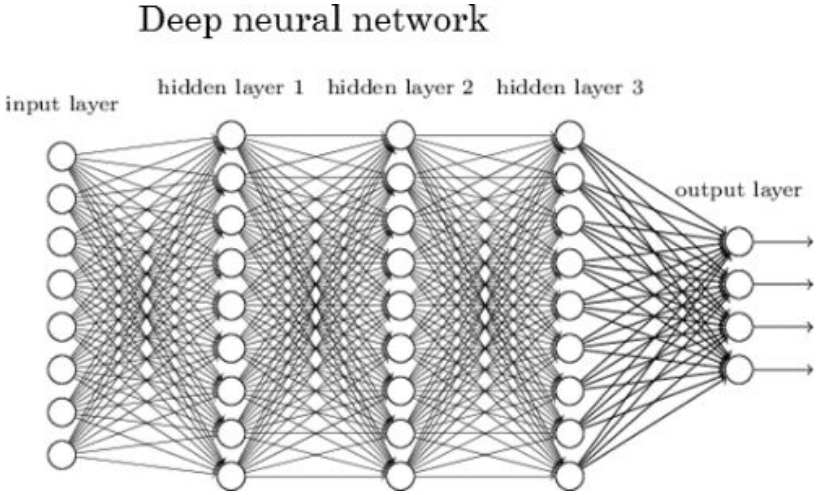


Q(table) graph

Q-Learning with Neural Networks

A neural network, in general, is a technology built to simulate the activity of the human brain – specifically, pattern recognition and the passage of input through various layers of simulated neural connections.

Many experts define deep neural networks as networks that have an input layer, an output layer and at least one hidden layer in between. Each layer performs specific types of sorting and ordering in a process that some refer to as “feature hierarchy.” One of the key uses of these sophisticated neural networks is dealing with unlabelled or unstructured data. The phrase “deep learning” is also used to describe these deep neural networks, as deep learning represents a specific form of machine learning where technologies using aspects of artificial intelligence seek to classify and order information in ways that go beyond simple input/output protocols.



Neural networks are fully capable of doing this on their own entirely.

While it is easy to have a 16x4 table for a simple grid world, the number of possible states in any modern game or real-world environment is nearly infinitely larger. For most interesting problems, tables simply don’t work. We instead need some way to take a description of our state, and produce Q-values for actions without a table: that is where neural networks come in. By acting as a function approximator, we can take any number of possible states that can be represented as a vector and learn to map them to Q-values.

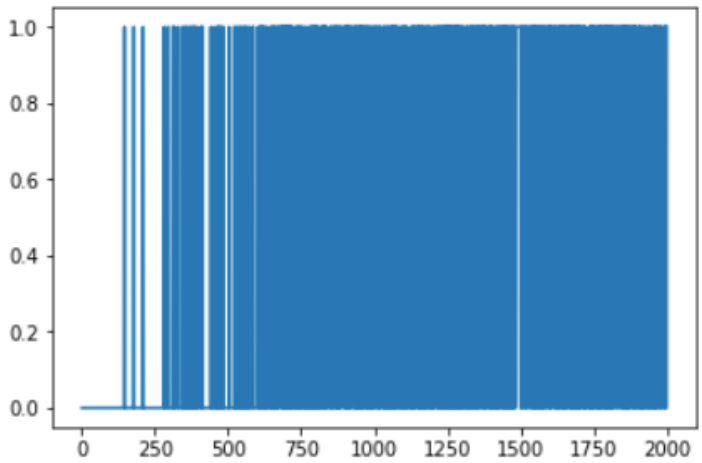
The High Level Implementation using Tensorflow:

1)In the case of the **FrozenLake** example, we will be using a **one-layer network which takes the state encoded in a one-hot vector (1x16),** and produces a vector of 4 Q-values, one for each action**. Such a simple network acts kind of like a glorified table, with the network weights serving as the old cells.**
The key difference is that we can easily expand the **Tensorflow network with added layers, activation functions, and different input types, whereas all that is impossible with a regular table.** The method of updating is a little different as well**. Instead of directly updating our table, with a network we will be using backpropagation and a loss function.** Our loss function will be sum-of-squares loss, where the difference between the current predicted Q-values, and the "target" value is computed and the gradients passed through the network**.** In this case, our Q-target for the chosen action is the equivalent to the Q-value computed**.**

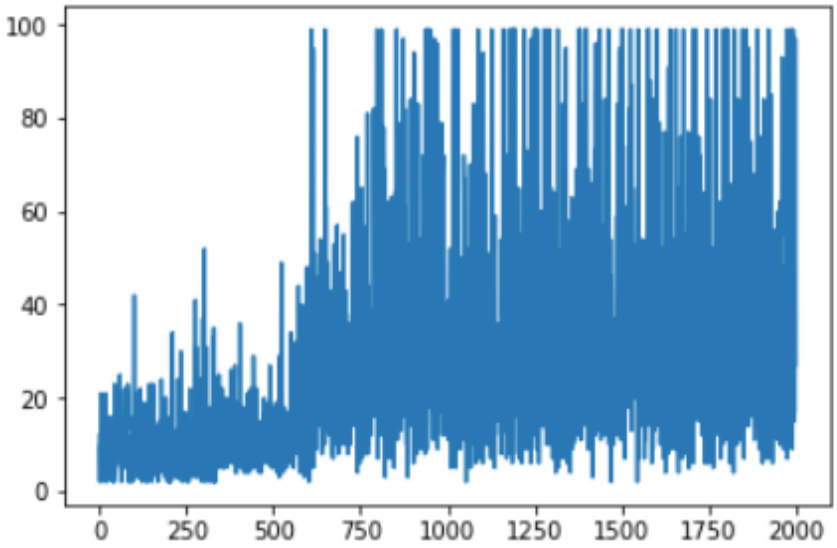
$$\text{Eq 1. } Q(s,a) = r+\gamma(\max(Q(s',a')))$$

$$\text{Eq2. Loss} = \sum(\text{Q-target} - Q)^2$$

Some statistics on network performance



We can see that the network beings to consistly reach the goal around the 650 episode mark.



It also begins to progress through the environment for longer than chance around the 650 mark as well.

Percent of succesful episodes: 0.424%

Conclusion

While the network learns to solve the FrozenLake problem, it turns out it doesn't do so quite as efficiently as the Q-Table. While neural networks allow for greater flexibility, they do so at the cost of stability when it comes to Q-Learning. There are a number of possible extensions to our simple Q-Network which allow for greater performance and more robust learning. Two tricks in particular are referred to as Experience Replay and Freezing Target Networks. Those improvements and other tweaks were the key to getting Atari-playing Deep Q-Networks.