Reinforcement Learning with Q-Learning

Deepanshu Yadav
UB Person No:
50321285
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
dyadav@buffalo.edu

Abstract

In this study, a reinforcement learning agent was built to navigate the classic 5x5 grid-world environment. The agent was made to learn an optimal policy through tabular Q-Learning which allowed it to take actions to reach a goal while avoiding obstacles. The environment and agent were built to be compatible with OpenAI Gym environments to make it run effectively on computationally-limited machines. Furthermore, two agents (random and heuristic agent) were provided as examples. The code for policy, updating Q-table and training process was implemented in this study. It was observed that by choosing hyperparamters wisely, the agent was able to learn quickly.

1 Introduction

1.1 Reinforcement Learning

Reinforcement learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment so as to maximize some reward. This is typically modeled as a Markov decision process (MDP), as illustrated in Figure 1.

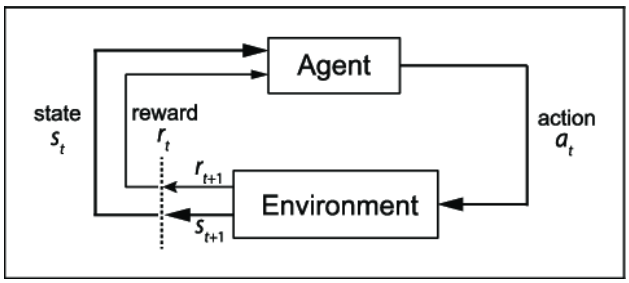

Fig 1: The canonical MDP diagram

An MDP is a 4-tuple *(S, A, P, R),* where
• *S* is the set of all possible states for the environment
• *A* is the set of all possible actions the agent can take
• $P = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the state transition probability function
• $R : S \times A \times S \rightarrow R$ is the reward function

Our task is to find a policy $\pi : S \rightarrow A$ which our agent will use to take actions in the environment which maximize cumulative reward, i.e.,

$$\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})$$

where $\gamma \in [0, 1]$ is a discounting factor (used to give more weight to more immediate rewards), s_t is the state at time step t, at is the action the agent took at time step t, and s_{t+1} is the state which the environment transitioned to after the agent took the action.

1.2 Q-Learning

Q-learning is an off-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

The 'Q' in Q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward. When q-learning is performed we create a *Q-table.*

Taking Action: Explore or Exploit

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as ***exploiting*** since we use the information, we have available to us to make a decision.
The second way to take action is to act randomly. This is called ***exploring***. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process.

We use a value iteration update algorithm to update our *Q-values* as we explore the environment's states:

$$Q^{new}(s_t, a_t) \leftarrow (1 - \alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Fig 2: Our update rule for Q-Learning

We are using our *Q-function* recursively to match (following our policy π) in order to calculate the discounted cumulative total reward. We initialize the table with all 0 values, and as we explore the environment (e.g., random actions), collect our trajectories, *[s_0, a_0, r_0, s_1, a_2, r_2, . . . , s_T , a_T , r_T]*, and use these values to update our corresponding entries in the *Q-table*.

2 Related Work

For this project, we are tasked with both implementing and explaining key components of the Q-learning algorithm. Specifically, a simple environment and a framework was provided to facilitate training, and we were responsible for supplying the missing methods of the provided agent classes. For this project, we implemented tabular Q-Learning, an approach which utilizes a table of Q-values as the agent's policy.

3 Environment

Reinforcement learning environments can take on many different forms, including physical simulations, video games, stock market simulations, etc. The reinforcement learning community (and, specifically, OpenAI) has developed a standard of how such environments should be designed, and the library which facilitates this is OpenAI's Gym.

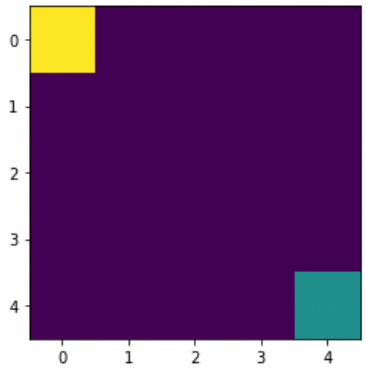

Fig 3: The initial state of our basic grid-world environment

The environment we provide is a basic deterministic n × n grid-world environment (the initial state for a 5 × 5 grid-world is shown in Figure 3) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.

The environment's state space will be described as an n × n matrix with real values on the interval [0, 1] to designate different features and their positions. The agent will work within an action space consisting of four actions: up, down, left, right. At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of +1 for moving closer to the goal and −1 for moving away or remaining the same distance from the goal.

4 Model Architecture

Overview: Reinforcement learning is a machine learning paradigm which focuses on how automated agents can learn to take actions in response to the current state of an environment so as to maximize some reward. This is typically modeled as a Markov decision process (MDP).

4.1 Markov Decision Processes:

A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

4.2 Implementation:

Implementing policy function:

Q-Learning is a process in which we train some function $Q_\theta : S \times A \rightarrow R$, parameterized by θ, to learn a mapping from state-action pairs to their *Q-value*, which is the expected discounted reward for following the following policy π_θ:

$$\pi(s_t) = argmax Q_\theta(s_t, \mathrm{a})$$

In words, the function Q_θ will tell us which action will lead to which expected cumulative discounted reward, and our policy π will choose the action a which, ideally, will lead to the maximum such value given the current state s_t.

Taking Action: Explore or Exploit

An agent interacts with the environment in 1 of 2 ways. The first is to use the q-table as a reference and view all possible actions for a given state. The agent then selects the action based on the max value of those actions. This is known as ***exploiting*** since we use the information, we have available to us to make a decision.
The second way to take action is to act randomly. This is called ***exploring***. Instead of selecting actions based on the max future reward we select an action at random. Acting randomly is important because it allows the agent to explore and discover new states that otherwise may not be selected during the exploitation process. You can balance exploration/exploitation using epsilon (ε) and setting the value of how often you want to explore vs exploit.

Updating the q-table

Originally, Q-Learning was done in a tabular fashion. Here, we would create an $|S| \times |A|$ array, our *Q-Table*, which would have entries $q_{i,j}$ where i corresponds to the *ith* state (the row) and j corresponds to the *jth* action (the column), so that if s_t is located in the *ith* row and at is the *jth* column, $Q(s_t,a_t) = q_{i,j}$. We use a value iteration update algorithm to update our *Q-values* as we explore the environment's states:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Fig 4: Our update rule for Q-Learning

We are using our *Q-function* recursively to match (following our policy π) in order to calculate the discounted cumulative total reward. We initialize the table with all 0 values, and as we explore the environment (e.g., random actions), collect our trajectories, $[s_0, a_0, r_0, s_1, a_2, r_2, \ldots, s_T, a_T, r_T]$, and use these values to update our corresponding entries in the *Q-table*.

The updates occur after each step or action and ends when an episode is done. Done in this case means reaching some terminal point by the agent. A terminal state for example can be anything like landing on a checkout page, reaching the end of some game, completing some desired objective, etc. The agent will not learn much after a single episode, but eventually with enough exploring (steps and episodes) it will converge and learn the optimal q-values.

Here are the 3 basic steps:
- Agent starts in a state (s_1) takes an action (a_1) and receives a reward (r_1)
- Agent selects action by referencing q-table with highest value (max) or by random (epsilon, ε)
- Update Q-values

We adjust our Q-values based on the difference between the discounted new values and the old values. We discount the new values using gamma and we adjust our step size using learning rate (α). Below are some references.

Learning Rate (*alpha* or α): It is defined as how much we accept the new value vs the old value. Above we are taking the difference between new and old and then multiplying that value by the learning rate. This value then gets added to our previous q-value which essentially moves it in the direction of our latest update.

Gamma: gamma or γ is a discount factor. It's used to balance immediate and future reward. From our update rule above you can see that we apply the discount to the future reward. Typically, this value can range anywhere from 0.8 to 0.99.

Reward: It is the value received after completing a certain action at a given state. A reward can happen at any given time step or only at the terminal time step.

The **max()** function helps the agent to always choose the state that gives it the maximum value of being in that state.

Implement the training algorithm

During training, the agent will need to explore the environment in order to learn which actions will lead to maximal future discounted rewards. Agent's often begin exploring the environment by taking random actions at each state. Doing so, however, poses a problem: the agent may not reach states which lead to optimal rewards since they may not take the optimal sequence of actions to get there. In order to account for this, we will slowly encourage the agent to follow its policy in order to take actions it believes will lead to maximal rewards. This is called exploitation, and striking a balance between exploration and exploitation is key to properly training an agent to learn to navigate an environment by itself.

To facilitate this, we have our agent follow what is called an ε-greedy strategy. Here, we introduce a parameter ε and set it initially to 1. At every training step, we decrease its value gradually (e.g., exponentially) until we've reached some predetermined minimal value (e.g., 0.1). In this case, we are annealing the value of ε exponentially so that it follows a schedule.

During each time step, we have our agent either choose an action according to its policy or take a random action by taking a sampling a single value on the uniform distribution over [0, 1] and selecting a random action if that sampled value is than ε otherwise taking an action following the agent's policy.

5 Results of Experimentation

For this project, we implemented tabular Q-Learning, an approach which utilizes a table of Q-values as the agent's policy.

Evaluation metrics: The epsilon decay and reward values have been evaluated with respect to episodes. The hyperparameters for the model, i.e. episodes, learning rate and gamma were set to get the optimum results by trial and error method. Epsilon decay and reward values were calculated for various values and the optimum ones were chosen as the final hyperparameters for the model. Figures below represent epsilon decay and reward plots for various values of hyperparameters.

5.1 Trial 1

- *Episodes: 1000, α: 0.5, γ: 0.88*

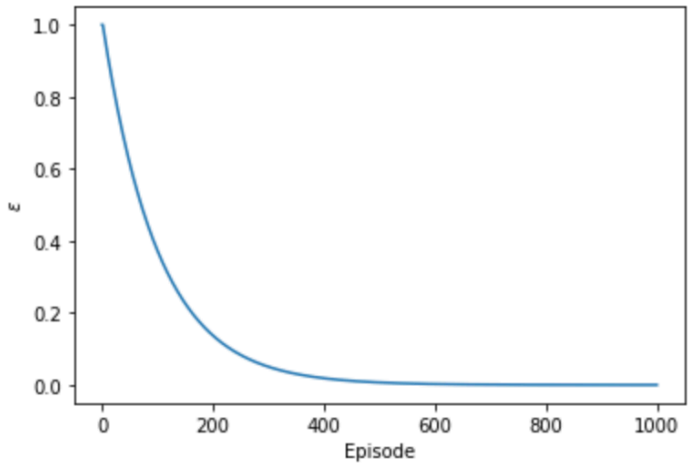

Fig 5: *Epsilon vs Episodes plot for Episodes: 1000, α: 0.5, γ: 0.88*

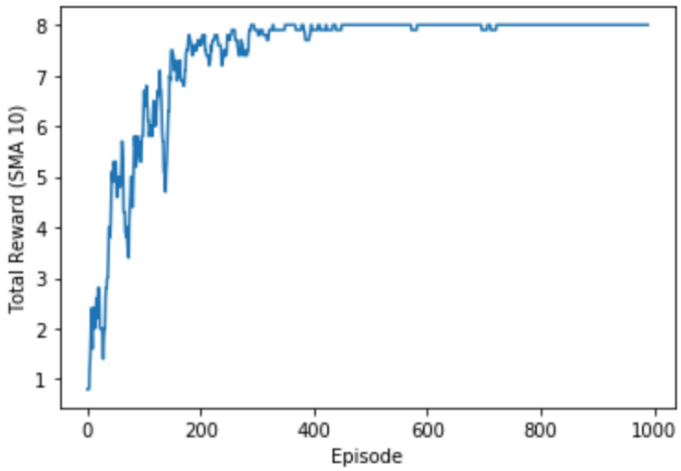

Fig 6: *Reward vs Episodes plot for Episodes: 1000, α: 0.5, γ: 0.88*

```
[[[ 5.33495132  3.69465923  5.33637896  3.69525702]
  [ 4.92770336  3.3362289   4.92755305  3.69599717]
  [ 4.46328505  1.82470804  2.41565475  2.11732075]
  [ 2.44350532  0.054092    0.97        1.4035699 ]
  [ 1.3502     -0.030608   -0.030608    0.        ]]

 [[ 4.47444045  2.91693489  4.92765578  2.99858227]
  [ 4.46329928  3.33456257  4.46244264  3.31225329]
  [ 3.93556736  2.54886122  2.91770768  2.85337523]
  [ 2.66675243  0.054092    0.875       1.34635448]
  [ 1.53265    -0.280608    0.         -0.28      ]]

 [[ 2.54395202  0.72395238  4.46312108  1.66224149]
  [ 3.92748253  2.83297961  3.93556736  2.92519637]
  [ 3.335872    2.46328958  3.28742862  2.44938848]
  [ 1.95277259  0.40021384  2.65437307  0.97840896]
  [ 1.87999865  0.153249    0.22541723  0.        ]]

 [[ 2.259242    0.143325    0.         -0.53      ]
  [ 1.19        0.035125    3.33587093  0.        ]
  [ 2.65263816  1.9245551   2.6544      1.93454298]
  [ 1.87480713  1.33536957  1.88        1.33299715]
  [ 1.          0.60400608 -0.14500005  0.64603545]]

 [[ 0.0302      0.          2.166149    0.        ]
  [ 0.067124    0.96682406  1.8571     -0.17      ]
  [ 0.5758998   0.66791223  1.87969604  0.405686  ]
  [-0.5        -0.17        0.99998474  0.29805   ]
  [ 0.          0.          0.          0.        ]]]
```

Fig 7: *Q-Table for Episodes: 1000, α: 0.5, γ: 0.88*

5.2 Trial 2

- *Episodes: 100, α: 0.9, γ: 0.99*

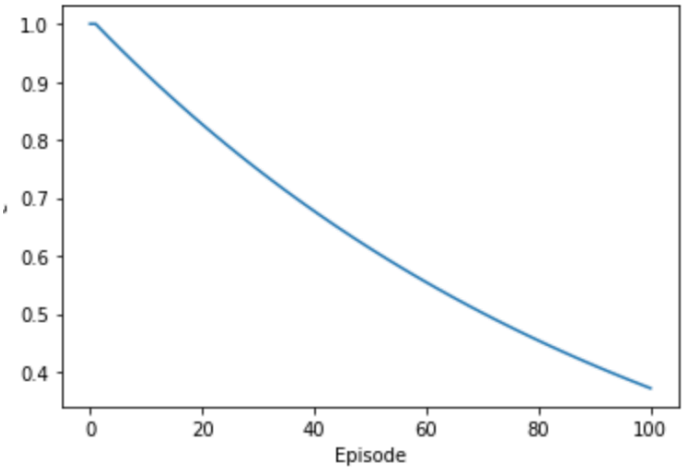

Fig 8: *Epsilon vs Episodes plot for r Episodes: 100, α: 0.9, γ: 0.99*

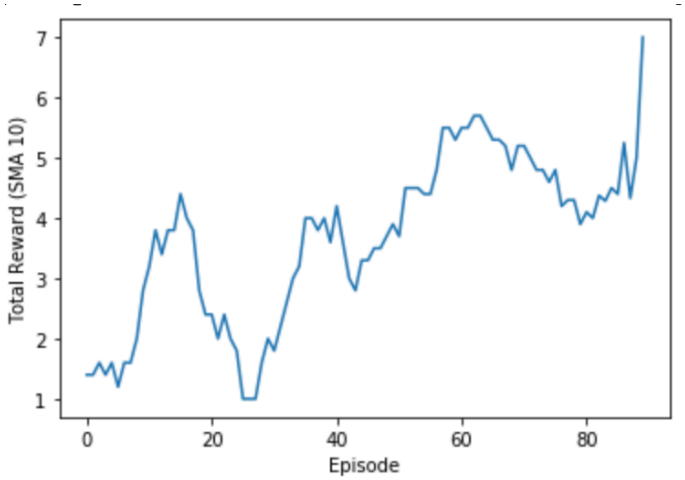

Fig 9: *Reward vs Episodes plot for Episodes: 100, α: 0.9, γ: 0.99*

```
[[[ 5.33634801  3.68064603  5.29402503  3.69459639]
  [ 4.92681049  3.23287929  4.55345088  3.29111364]
  [ 4.43122015  2.55345341  0.          1.2646669 ]
  [ 0.          0.          0.          0.        ]
  [ 0.         -0.9         0.          0.        ]]

 [[ 4.83921858  3.63590181  4.92767217  3.28353943]
  [ 4.46327378  3.32186932  4.30664154  3.21708306]
  [ 3.93475557  2.66731365  2.4269256   2.87800741]
  [ 1.7028      0.          0.9         1.18132702]
  [ 0.9        -0.9         0.          0.        ]]

 [[ 2.61063288  2.21694301  4.45429093  1.72876295]
  [ 3.67517381  2.92181318  3.93553869  2.75633652]
  [ 3.33583992  2.450145    1.8764568   2.38612313]
  [ 0.          0.          0.9999      0.58894667]
  [ 0.         -0.9         0.          0.        ]]

 [[ 1.85436    -0.11592     0.         -0.9999    ]
  [ 1.88109674  2.04882048  3.31148655 -0.1872    ]
  [ 1.40207124  1.93545792  2.65436361  1.67320445]
  [ 1.87999088 -0.1080792   1.7028      0.56381564]
  [ 0.99        0.          0.          0.        ]]

 [[ 0.          0.          0.99        0.        ]
  [-0.99        1.6603684   0.         -0.11592   ]
  [-0.58792875  1.2529699   0.         -0.9       ]
  [-0.9         0.58895046  0.99999    -0.9       ]
  [ 0.          0.          0.          0.        ]]]
```

Fig 10: *Q-Table for Episodes: 100, α: 0.9, γ: 0.99*

5.3 Trial 3

- *Episodes: 1000, α: 0.1, γ: 0.90*

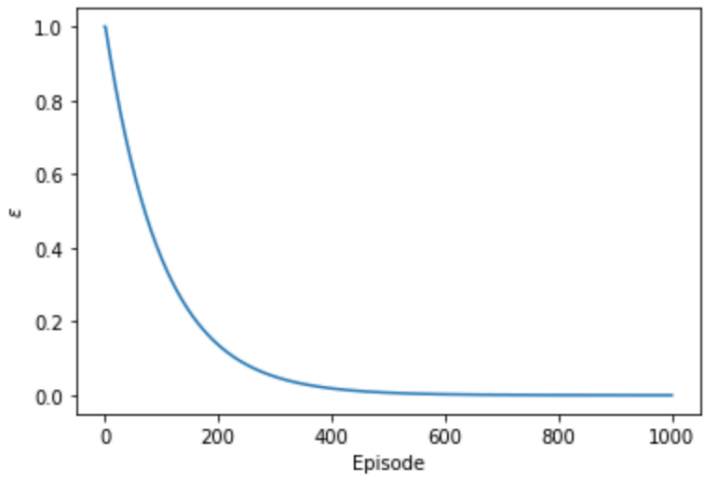

Fig 11: *Epsilon vs Episodes plot for Episodes: 1000, α: 0.1, γ: 0.90*

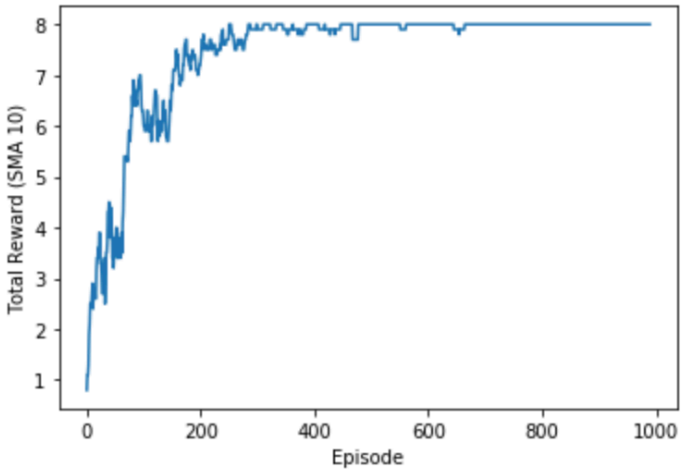

Fig 12: *Reward vs Episodes plot for Episodes: 1000, α: 0.1, γ: 0.90*

```
[[[ 3.59884589  2.41716718  5.6953279   2.09622864]
  [ 5.217031    1.72091523  2.08725366  2.71635259]
  [ 2.33129362  0.          0.20074312 -0.05782969]
  [ 0.2360575   0.          0.         -0.09019   ]
  [ 0.          0.          0.          0.        ]]

 [[ 1.09121924  0.10535386  4.58792549  0.68073287]
  [ 4.68559     2.88629034  2.19352611  1.74733281]
  [ 2.92608322 -0.09816609  0.3979036   0.21089807]
  [ 0.4180969  -0.17925688  0.1         0.        ]
  [ 0.10981     0.          0.         -0.080632  ]]

 [[ 0.2978119   0.57590635  2.83202283 -0.16750592]
  [ 4.0951      2.17544794  2.28406095  0.31377001]
  [ 3.2646129   0.02494543  0.65062093  0.102049  ]
  [ 0.2173141  -0.1613737   0.58462966 -0.0829    ]
  [ 0.4290238  -0.25222249  0.         -0.091     ]]

 [[ 0.11188472  0.05213047  3.16606394  0.        ]
  [ 1.71742336  1.3728      3.439       0.80673701]
  [ 2.71        0.80841525  1.30877369  1.09725066]
  [ 0.16712321 -0.07561     1.12467754  0.25211115]
  [ 0.61257951  0.         -0.150949    0.        ]]

 [[-0.1         0.25047279  0.          0.        ]
  [-0.1         0.32335703  1.8801118  -0.26969087]
  [ 0.38357233  0.89706274  1.9         0.05209764]
  [-0.12023441 -0.17142168  1.          0.28379056]
  [ 0.          0.          0.          0.        ]]]
```

Fig 13: *Q-Table for Episodes: 1000, α: 0.1, γ: 0.90*

From the results obtained from different trials, we can observe that the agent's learning has been best when number of episodes are 1000, the learning rate is 0.1 and gamma value is 0.90.

6 Conclusion

In this study, a reinforcement learning agent was built to navigate the classic 5x5 grid-world environment. The agent was made to learn an optimal policy through tabular Q-Learning which allowed it to take actions to reach a goal while avoiding obstacles. The environment and agent were built to be compatible with OpenAI Gym environments to make it run effectively on computationally-limited machines. Furthermore, two agents (random and heuristic agent) were provided as examples. The code for policy, updating Q-table and training process was implemented in this study. It was observed that the agent's learning has been best when number of episodes are 1000, the learning rate is 0.1 and gamma value is 0.90.

R e f e r e n c e s

[1] https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/
[2] https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56
[3] https://arxiv.org/abs/1509.06461
[4] https://deeplizard.com/learn/video/qhRNvCVVJaA
[5] https://deeplizard.com/learn/video/QK_PP_2KgGE
[6] https://deeplizard.com/learn/video/mo96Nqlo1L8