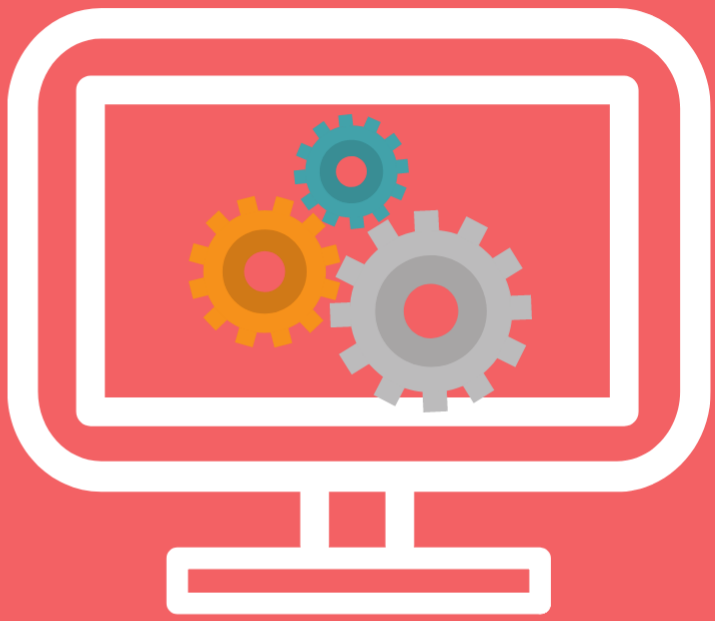
Reinforcement Learning

Q-Learning

By Group A

Andres Behnsen, Amin Abaza, Jonas Hellevang, Adelina Akhsanova, Ayush Kejriwal and Dwight Alexander Schirra



Prof. Antonio Rodriguez Armas

1. What is Q-learning?

Within reinforcement learning there are two main algorithm methods; model-based and model-free. Model-based algorithms utilize reward functions to progressively improve the model, on the other hand model-free algorithms disregard the normal reward functions and estimate their own value functions for an unknown Markov decision process, based on experience. Examples of these different type of algorithm are Monte-Carlo Learning, Temporal-Difference Learning, and Q-Learning.

The most popular of these methods is Q-Learning and like other model-free algorithms is characterized as off-policy learning. This means that the algorithm finds the best possible action considering its current position. Through this method an agent's objective is to obtain the maximum number of points, and Q-learning attempts to learn how many points it is expected to get if it were to *behave perfectly*. It does this by trying things out (either randomly or following some set of rules), receiving points, and adjusting its expectations based on the points it immediately received and its expected best case scenario for the number of points it can receive in the future.

A key idea behind Q-learning is that an agent learns about how good actions are under perfect behavior without having to behave perfectly.

The most classic examples of Q-learning are used to teach agents to navigate a certain path of objects with rewards being given for achieving a certain completion of the path. This simple process can be extrapolated and built upon to accomplish other complex tasks that will be discussed later.

2. Applying RL and Q-learning to your problem

In order for any problem to constitute a real Reinforcement learning or Q-Learning application it needs to comply with certain things:

1. Trial and error environment —> where the same scenario plays and replays.
2. Delayed rewards —> after completing a go it gets fed a reward based on certain parameters previously inputed by the user.
3. Your problem is a control problem —> it needs a steady, guiding hand in order to better run.
4. Can be modeled as a Markov Decision Process (MDP) —> this means that it has to follow a decision tree implementation and needs to be referred to the start over and over again.
5. A simulated environment —> the same environment repeats itself so the agent can learn based on its advance.

This 5 steps determine if a Reinforcement Learning can be applied to your model. Remember to always go through this checklist. In case that it doesn't complete the checklist it is not recommendable to keep pursuing an RL algorithm since more suited options would be available.

3. Formula and variables

Q-function uses Bellman equation and takes as inputs actions(a) and state(s):

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

- New Q Value for that action and state
- Current Q Values
- Learning Rate
- Reward for taking that action at that state
- Discount rate
- Maximum expected future reward given the new state(s') and all possible actions at that new state

In the formula we can see how the performance of the algorithm changes with different values of the variables:

1. Learning Rate (also called a step size) —> defines how much last acquired information rewrites information gained before. Coefficient equals 0 means agent does not consider new information and uses only previously learnt information, whereas 1 means agent dumps previous information and considers only info acquired with the last step. It is used when the environment is absolutely deterministic.

2. Discount rate —> defines significance of future rewards. With coefficient of 0 the agent considers only the closest rewards whereas 1 makes it "sagacious" – it counts future rewards.

3. Initial Values —> can be zeros and then the agent does not have any knowledge about the environment. In this case its actions will be random. If initial values are high the choice of some options will be higher than others. It helps to navigate agent to the "right" way.

4. Q-Learning Algorithm

An agent tries different actions in different states and stores the results in a Q-table. Iterating process makes recorded in the table values more precise.

1. Initialize Q-Table —> Q-Table is a table where columns is number of actions and rows is number of states. In the Q-table we calculate the maximum expected future rewards at each state for all actions. Actions can be for example go up, go down, go to the left, go to the right. In our example all the values (Q-values) in the table are zeros that is why at the beginning agent will start to act randomly.

Action/States	A	B	C	D
a	0	0	0	0
b	0	0	0	0
c	0	0	0	0
d	0	0	0	0

2. Choose an action and perform it —> Agent chooses action and implements it as long as it defined in the code. For that there are many strategies which can be implemented. If the agent does not have any knowledge about the environment it chooses random action. Values are updated too based on its performance.

3. Measure Reward and Update Table —> After each iteration we see an outcome and update the table. Algorithm will choose and perform action, measure reward and update Q-table again and again and after some time of iterations Q-table will be filled.

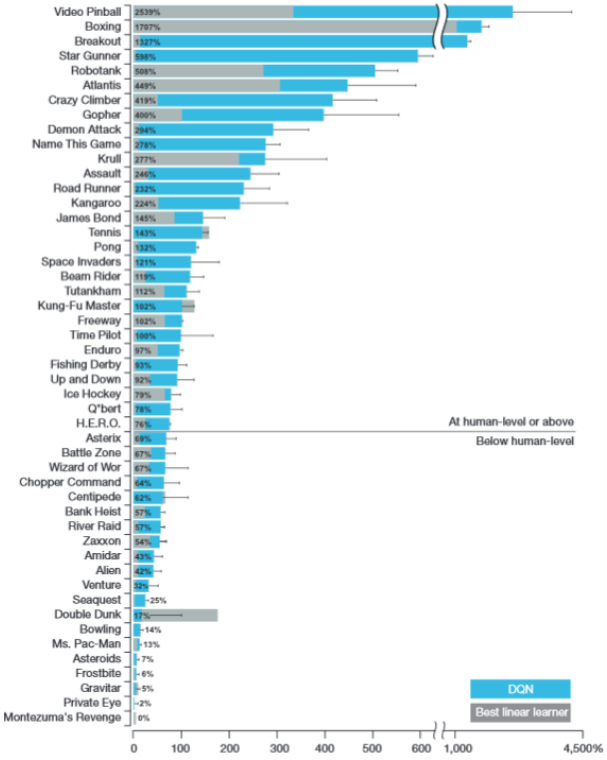
Action/States	A	B	C	D
a	**1**	0	0	0
b	0	0	0	0
c	0	0	0	0
d	0	0	0	0

5. Q-Learning Use Cases

- Web System configuration —> On a web system there exist more than 100 parameters to configure and tune. This requires a person that has to run an extensive amount of trail and error tests in order to find the best combination. This is where Q-Learning can help show how to do autonomic reconfiguration of parameters in multi-tier web systems in VM-based dynamic environments. This helps take away the human trial and error become autonomous and faster by using Q-Learning and a reward function based on the number of jobs on the queued and the priority of those jobs. Since this an evolving discipline that offers a lot of trial and error is perfect for Reinforcement Learning and Q-Learning.

- Personalized Recommendations —> Nowadays people's attention span is very limited, they become bored supremely fast. This has been posting a challenge for the news industry and the news they feed onto us. The specific problem is Click Through Rate cannot reflect the retention rate of users. To tackle this problem they decided to split into four categories features. This 4 categories would provide them a more in depth analysis for each category. This 4 categories were later fed into a Deep Q-Network to calculate the resulting Q-value. A list of news were chosen by its Q-value to feed each category and the user's click on the news was added as part of the reward the agent received.

- Gaming/E-Sports —> The E-Sports and gaming community have been thrust into hyperdrive on the last years. Since the purchase of several E-Sports teams by Google, lots of software developers and big companies have been exploring the waters of E sports and Gaming. The idea behind this is that they can use several of the algorithms used for gaming into other use cases like autonomous driving in the case of google. Where Q-Learning fits better is in the learning part of the game. Upon millions and millions of training simulations machines are thrust into battle versus gaming teams. After learning and relearning it usually performs and in some cases beats the actual gamers. This is done via Reinforcement Learning and Q-Learning.

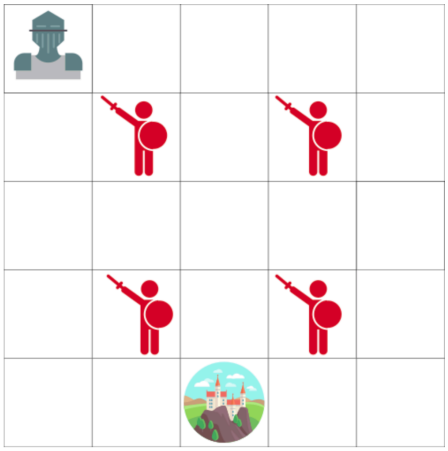


6. Q Learning Example

To further illustrate what is Q-Learning here is an example, The Knight and the Princess game. This game is a typical Q-Learning example, the objective of this game is to get from the top left corner to the castle in order to save the princess without landing in any of the guardian spots. This is a 5x5 board, the player or agent can only move one step at a time. The enemy; the guardians cannot move, they are static.

This type of problem can be evaluated using a point scoring system. This would be an example of an initial point system:

- +100 points for reaching the castle, you win and the simulation ends.
- -100 points for landing in a guardian and the simulation ends.
- -1 point for step he takes; this encourages our actor to go faster.



Now that we have our the scoring system set up how do we get actually get to the castle. To make it more clear let's map out in green and red where you would be able to go. This map would show green spaces pointing where the agent can move and red spaces where the simulation would end by stepping into a guardians box.

Even after mapping it would seem very simple, just tell the agent to take all the green tiles until the castle. The problem with this interpretation is

that the agent can easily be caught in an infinity loop going around one of the guardians.

As it turns out is not so simple making the knight go to the castle. As previously mentioned in this document the usage of the Q-Table is needed. In this case the Q-

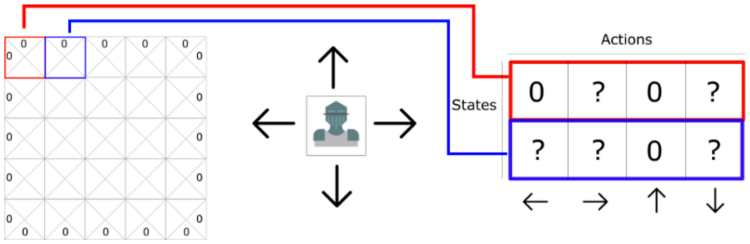
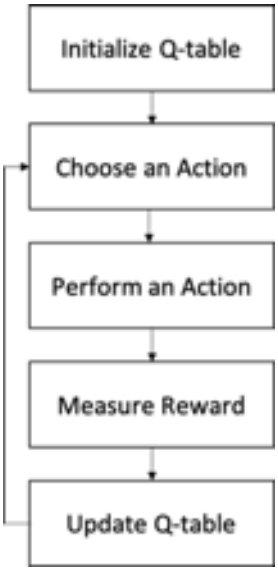


Table is going to be 4 columns one for each side the knight can move. In the case of the corners there is only two and boundaries are three possible movements. Nonetheless the knight has his range of movement on the Q-Table.

Now that we have our Q-Table, its time to run the Q-Learning Algorithm process. This process is a constant trial and error that registers each movement on the Q-Table. Since all movements are recorded we add the total of points per run. As we can see in the picture below the Q-Learning Algorithm Process is a loop that runs itself as many times as the user sets it to, on a given interval of time or forever. After certain amounts of runs the knight will get further and further down the board until it reaches the castle. This would mean that the knight is “learning” and that is how Q-Learning works.



7. Bibliography

- Garychl, Applications of Reinforcement Learning in Real World, August 2, 2018

https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12

- Andre Violante, Simple Reinforcement Learning Q Learning, March 18, 2019

https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

- freeCodeCamp, Diving deeper into Reinforcement Learning with Q-Learning, April 10, 2018.

https://www.freecodecamp.org/news/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe/

- Chakrit Yau, Reinforcement Learning Q Learning with Decision Trees , November 20, 2018

https://towardsdatascience.com/reinforcement-learning-q-learning-with-decision-trees-ecb1215d9131

- Chathurangi Shyalika, November 15, 2019

https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c

- Teknomo, Kardi. Q-Learning by Examples, 2005

https://people.revoledu.com/kardi/tutorial/ReinforcementLearning/Q-Learning-Algorithm.htm

- Satwik Kansal & Brendan Martin, Reinforcement Q-Learning from Scratch in Python with OpenAI Gym

https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/