

★ Get unlimited access to all of Medium for less than \$1/week. [Become a member](#)



An introduction to Q-Learning: reinforcement learning



Akshay Lamba · [Follow](#)

Published in [We've moved to freeCodeCamp.org/news](#)

6 min read · Sep 3, 2018

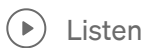


Photo by [Daniel Cheung](#) on [Unsplash](#).

This article is the second part of my “Deep reinforcement learning” series. The complete series shall be available both on [Medium](#) and in videos on [my YouTube channel](#).

In the [first part of the series](#) we learnt the **basics of reinforcement learning**.

Q-learning is a values-based learning algorithm in reinforcement learning. In this article, we learn about Q-Learning and its details:

- What is Q-Learning ?
- Mathematics behind Q-Learning
- Implementation using python

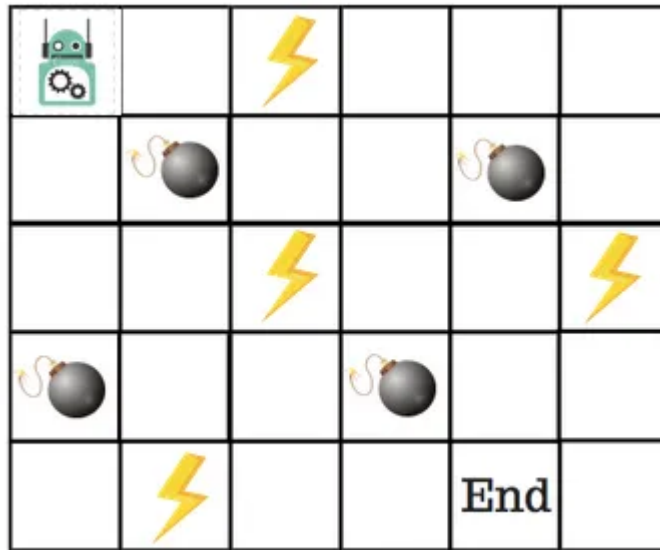
Q-Learning — a simplistic overview

Let's say that a **robot** has to cross a **maze** and reach the end point. There are **mines**, and the robot can only move one tile at a time. If the robot steps onto a mine, the robot is dead. The robot has to reach the end point in the shortest time possible.

The scoring/reward system is as below:

1. The robot loses 1 point at each step. This is done so that the robot takes the shortest path and reaches the goal as fast as possible.
2. If the robot steps on a mine, the point loss is 100 and the game ends.
3. If the robot gets power ⚡, it gains 1 point.
4. If the robot reaches the end goal, the robot gets 100 points.

Now, the obvious question is: **How do we train a robot to reach the end goal with the shortest path without stepping on a mine?**



So, how do we solve this?

Introducing the Q-Table

Q-Table is just a fancy name for a simple lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state.



There will be four numbers of actions at each non-edge tile. When a robot is at a state it can either move up or down or right or left.

So, let's model this environment in our Q-Table.

In the Q-Table, the columns are the actions and the rows are the states.

Actions :	↑	→	↓	←
Start				
Nothing / Blank				
Power				
Mines				
END				

Each Q-table score will be the maximum expected future reward that the robot will get if it takes that action at that state. This is an iterative process, as we need to improve the Q-Table at each iteration.

But the questions are:

- How do we calculate the values of the Q-table?
- Are the values available or predefined?

To learn each value of the Q-table, we use the **Q-Learning algorithm**.

Mathematics: the Q-Learning algorithm

Q-function

The **Q-function** uses the Bellman equation and takes two inputs: state (s) and action (a).

$$Q^{\pi}(s_t, a_t) = \underline{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | s_t, a_t]$$

Q-Values for the state given a particular state

Expected discounted cumulative reward

Given the state and action

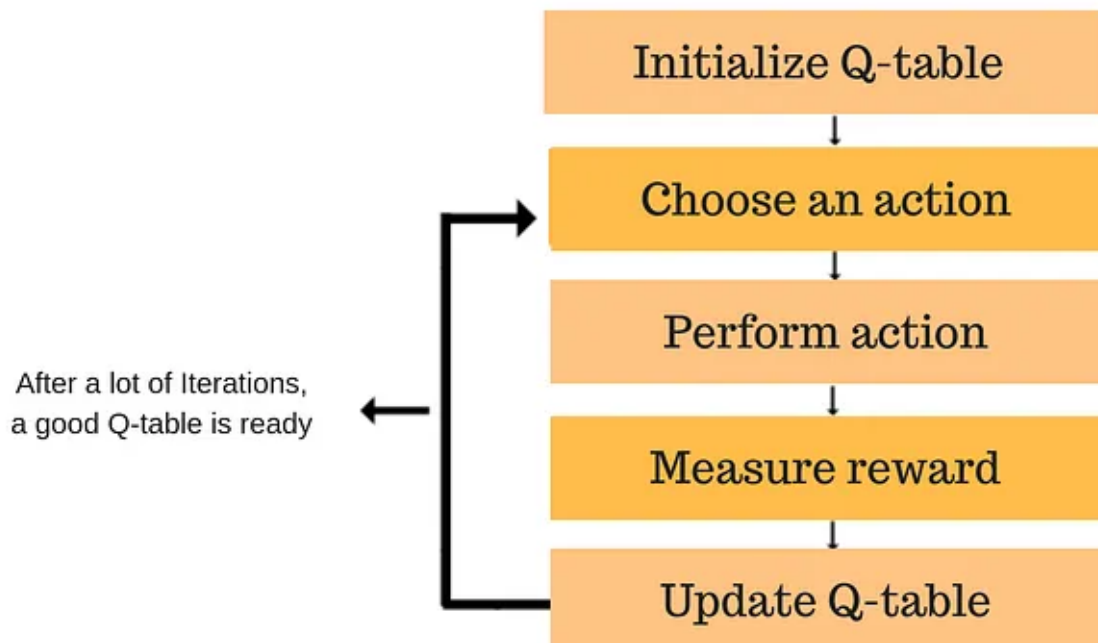
Using the above function, we get the values of Q for the cells in the table.

When we start, all the values in the Q -table are zeros.

There is an iterative process of updating the values. As we start to explore the environment, the Q -function gives us better and better approximations by continuously updating the Q -values in the table.

Now, let's understand how the updating takes place.

Introducing the Q-learning algorithm process



Each of the colored boxes is one step. Let's understand each of these steps in detail.

Step 1: initialize the Q-Table

We will first build a Q-table. There are n columns, where n = number of actions. There are m rows, where m = number of states. We will initialise the values at 0.

Actions :				
Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

					
					
					
					
				End	

In our robot example, we have four actions ($a=4$) and five states ($s=5$). So we will build a table with four columns and five rows.

Steps 2 and 3: choose and perform an action

This combination of steps is done for an undefined amount of time. This means that this step runs until the time we stop the training, or the training loop stops as defined in the code.

We will choose an action (a) in the state (s) based on the Q-Table. But, as mentioned earlier, when the episode initially starts, every Q-value is 0.

So now the concept of exploration and exploitation trade-off comes into play. [This article has more details.](#)

We'll use something called the **epsilon greedy strategy**.

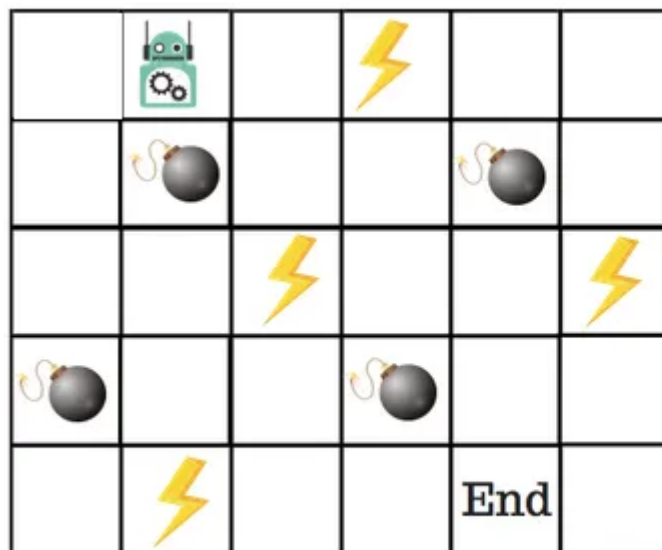
In the beginning, the epsilon rates will be higher. The robot will explore the environment and randomly choose actions. The logic behind this is that the robot does

not know anything about the environment.

As the robot explores the environment, the epsilon rate decreases and the robot starts to exploit the environment.

During the process of exploration, the robot progressively becomes more confident in estimating the Q-values.

For the robot example, there are four actions to choose from: up, down, left, and right. We are starting the training now — our robot knows nothing about the environment. So the robot chooses a random action, say right.



Actions : ↑ → ↓ ←

Start	0	0	0	0
Nothing / Blank	0	0	0	0
Power	0	0	0	0
Mines	0	0	0	0
END	0	0	0	0

We can now update the Q-values for being at the start and moving right using the Bellman equation.

Steps 4 and 5: evaluate

Now we have taken an action and observed an outcome and reward. We need to update the function $Q(s,a)$.

$$\text{New } Q(s, a) = Q(s, a) + \alpha [R(s, a) + \gamma \max_{a'} Q'(s', a') - Q(s, a)]$$

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

In the case of the robot game, to reiterate the scoring/reward structure is:

- **power** = +1
- **mine** = -100
- **end** = +100

New $Q(\text{start}, \text{right}) = Q(\text{start}, \text{right}) + \alpha [\text{some ... Delta value}]$

Some ... Delta value = $R(\text{start}, \text{right}) + \max(Q'(\text{nothing}, \text{down}), Q'(\text{nothing}, \text{left}), Q'(\text{nothing}, \text{right})) - Q(\text{start}, \text{right})$

Some ... Delta value = $0 + 0.9 * 0 - 0 = 0$

New $Q(\text{start}, \text{right}) = 0 + 0.1 * 0 = 0$

actions.

Next time we'll work on a deep Q-learning example.

Until then, enjoy AI 😊.

Important: As stated earlier, this article is the second part of my “Deep Reinforcement Learning” series. The complete series shall be available both in articles on [Medium](#) and in videos on [my YouTube channel](#).

If you liked my article, **please click the** 🙌 to help me stay motivated to write articles. Please follow me on **Medium** and other social media:



Follow me on Instagram



Follow me on Twitter



Follow our Channel

If you have any questions, please let me know in a comment below or on [Twitter](#).

Subscribe to [my YouTube channel](#) for more tech videos.

Machine Learning

Artificial Intelligence

Deep Learning

Reinforcement Learning

Tech



Follow

Written by Akshay Lamba

516 Followers · Writer for [We've moved to freeCodeCamp.org/news](#)

Startup, Web, Mobile Dev, AI, ML :) Instagram : <https://www.instagram.com/buildlikelamba> # LinkedIn : <https://www.linkedin.com/in/iamadl/>

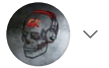
More from Akshay Lamba and We've moved to freeCodeCamp.org/news



Open in app ↗



Search Medium



 Akshay Lamba in We've moved to freeCodeCamp.org/news

Get to know TensorFlow.js in 7 minutes

And learn how you can run ML/DL models directly in the browser

7 min read · Jul 26, 2018




723



4





 TK in [We've moved to freeCodeCamp.org/news](https://freeCodeCamp.org/news)

Learning Python: From Zero to Hero

This post was originally published at TK's Blog.

11 min read · Sep 30, 2017



67K



136



 Peter Gleeson in [We've moved to freeCodeCamp.org/news](#)

An A-Z of useful Python tricks

Python is one of the world's most popular, in-demand programming languages

9 min read · Aug 28, 2018



29K



57



 Akshay Lamba in [Towards Data Science](#)

Introduction to ML5.js

A Beginner's Friendly Machine Learning for the Web.

5 min read · Aug 8, 2018



236



1



[See all from Akshay Lamba](#)

[See all from We've moved to freeCodeCamp.org/news](#)

Recommended from Medium



Waleed Mousa in Artificial Intelligence in Plain English

Building a Tic-Tac-Toe Game with Reinforcement Learning in Python: A Step-by-Step Tutorial

Welcome to this step-by-step tutorial on how to build a Tic-Tac-Toe game using reinforcement learning in Python. In this tutorial, we will...

9 min read · Mar 13



10



1



 Mehul Gupta in Data Science in your pocket

Training OpenAI gym environments using REINFORCE algorithm in reinforcement learning

Policy gradient methods explained with codes

8 min read · Mar 26



Lists



Predictive Modeling w/ Python

18 stories · 258 saves



Natural Language Processing

494 stories · 128 saves



AI Regulation

6 stories · 77 saves



Practical Guides to Machine Learning

10 stories · 269 saves



Eligijus Bujokas in Towards Data Science

The Values of Actions in Reinforcement Learning using Q-learning

The Q-learning algorithm implemented from scratch in Python

★ · 10 min read · Feb 14



18



 Samandar Xamidov

Cart Pole Gym using Reinforcement Learning

Welcome to CartPole prooject!

4 min read · Feb 16

 216  1

 Cybernova Blog in Javarevisited

A Detailed Introduction to Reinforcement Learning (RL) | Cybernova

Access to humongous amounts of data and the availability of enormous computational power have led us to explore various techniques to draw...

11 min read · Jul 9

 54 



Timothe Boulet in InstaDeep

Deep Reinforcement Learning for Network Design in Marine Transportation

Complex constrained optimization problems such as Marine Transportation's Network Design can be tackled with RL methods with many...

18 min read · Mar 6



52



See more recommendations