Hyperiondev

TASK

Reinforcement Learning

Visit our website

Introduction

WELCOME TO THE REINFORCEMENT LEARNING TASK!

This task describes the concept of reinforcement learning including Markov Decision Processes and Q-Learning.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

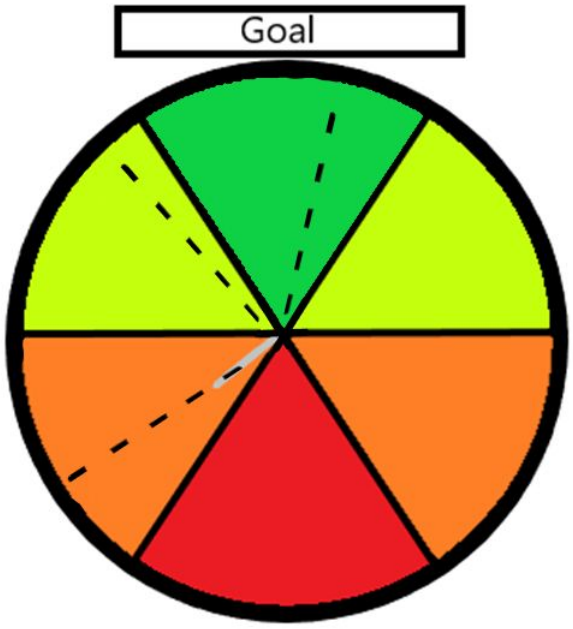
The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

WHAT IS REINFORCEMENT LEARNING?

Reinforcement learning is the type of learning that humans are naturally most familiar with — learning through interaction with our environment. For example, a toddler learning to walk or a school child learning to skip with a rope. In the first example, the child might receive praise when they manage to walk to their parents and so they learn that if they move in this way they get praise, but if they fall then it hurts. The child learning to skip has to learn to gauge how soon to jump so the rope doesn't hit their feet, but how long to wait so the rope doesn't hit their head. The important aspects of these processes are that the person interacts with their environment and receives reward or punishment which encourages them to take the behaviour in future with the most reward.

Let's look at another simple example in more detail. Imagine you were on a soccer field and needed to work out which was the best direction to kick the ball in, such that you would score a goal. If you look around you can see where the goal is, and so intuitively you know you should kick the ball in that direction, as there is a good chance that the ball might go in, as opposed to kicking the ball in the opposite direction, which would give you a zero chance of getting the ball in.



Now imagine that you are blindfolded and you don't know where the goal is, you just have an initial starting position in which you face (eg. the short grey line in the picture above). On each attempt, you will return to that starting position before turning to whichever direction you decide to kick in next. Since you have no idea where the goal is, you simply kick the ball forwards (in the orange region). It

doesn't go into the goal, so you return to the initial position and get zero points. You turn a little bit to the right and kick again into the yellow region — again the ball doesn't go in, so you get zero points. On your next turn, you turn a little more to the right than before and kick again into the green region, and the ball goes in — so you get a point! If you were to remain there on the field all day kicking the ball you would eventually work out exactly where the green region was, ie. the region with a high probability of getting the ball in, as well as working out the best direction to face such that when kicking the ball in that direction you would have the highest likelihood of getting the ball into the goal, and getting a point, ie. straight into the goal. Thus, you are able to learn which direction to kick the ball, without being able to see or know where the goal is.

When the person started kicking the ball, they had no idea which directions would have a high likelihood of a goal, and which a low likelihood. It was only through interaction with the environment that they were able to learn which directions were more likely to give a point or reward. If we were to describe this example in reinforcement learning terms we would call the person the **agent**, the field with the goal the **environment**, the position the agent faced in the environment the **state** and the points are the **reward.** By repeatedly kicking the ball in different directions, the person was able to maximise this reward.

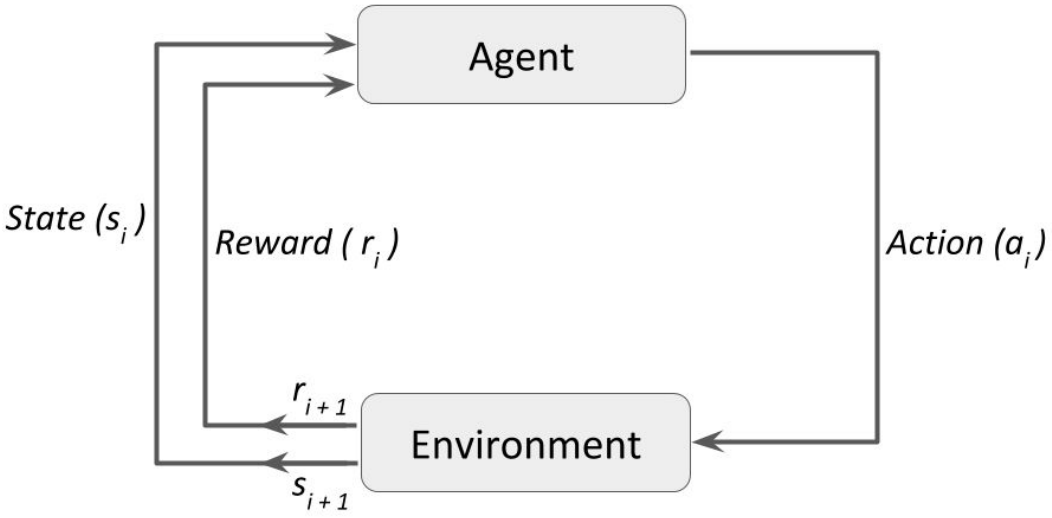
COMPONENTS OF REINFORCEMENT LEARNING

More formally, reinforcement learning is the process of training an agent to maximise the reward through interaction with its environment using a specific set of **actions**. Let's define each of these components of reinforcement learning:

- **Agent**: An entity that interacts with the environment by taking actions and receiving both a new state and reward from the environment as a result of the action. The agent tries to maximise its reward. In terms of machine learning, the agent is the model we train.
- **Environment**: A scenario or task in which an agent can take a set of actions, and which supplies a state and reward to the agent.
- **Action (a)**: One of a set of possible actions/moves that the agent can take to interact with its environment, allowing the agent to change its state within the environment. Each action yields a reward from the environment.
- **State (s)**: The immediate scenario within the environment that the agent is faced with.
- **Reward (r)**: A numerical value sent to the agent by the environment as a result of the action the agent just took.

*For your interest: for extensive definitions relating to all aspects of reinforcement learning, have a look at **this** blog.*

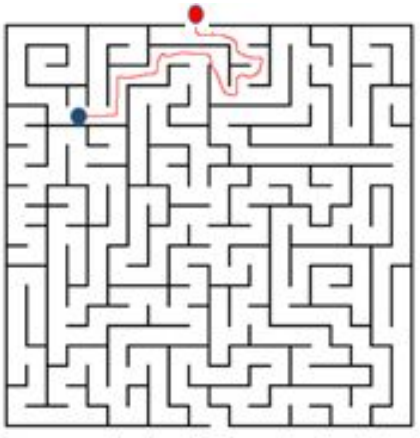
The interactions of these different components are summarised through the following diagram:



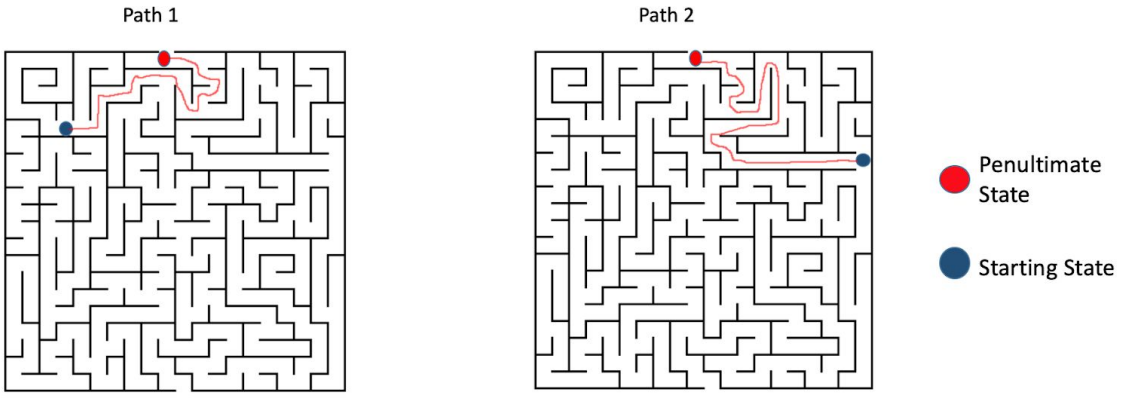
(***Source***)

MARKOV DECISION PROCESSES

Let's consider an example in which your goal is to exit a maze, starting at the blue dot (in the diagram below) and ending at the red dot. In this example, to get to the state of the red dot (exited the maze), a specific chain of states and actions from those states needed to occur.



However, considering all the states that have come before the current one in order to decide which action to take next (before reaching the end state) can be very computationally expensive.



(***Source***)

If we consider two different starting points in the maze, where both cases have achieved the penultimate state, we see that from this state, the action that needs to be taken in both states is to move up to exit the maze. Thus, the ultimate state only depends on the penultimate state and the transition from one state to the other, and not on the sequence of states that occurred before the penultimate state. We call this a **Markov state**.

Markov states are defined such that each state only depends on the state before it and the transition (which consists of both the action and reward) from the previous state to the current state. In reinforcement learning, we consider all the states to be Markov states, and hence reduce the computational cost of calculating which action to take next. The process of determining which action to take in this way is called the Markov decision process.

THE BELLMAN EQUATION

The aim of the agent is to maximise the cumulative reward it receives, which is often referred to as the quality value (or 'Q-value'). Thus, at each state, the agent needs to take action such that its overall sequence of actions will result in the highest Q-value. This means that we need a measure of the value of each action at each state.

An equation called the **Bellman equation** describes this, which is written mathematically as follows:

$$Q(s,a) = r(s,a) + \gamma * \max(Q(s',a'))$$

In other words, this equation tells us that the value of an action from a particular state is the value of the reward we immediately receive for this action as well as the reward of the most optimal actions taken in the next state we move to (reduced by a factor of γ, because we have to wait for this reward). γ is called the **discount factor**, and this controls how much the future rewards contribute to the current reward.

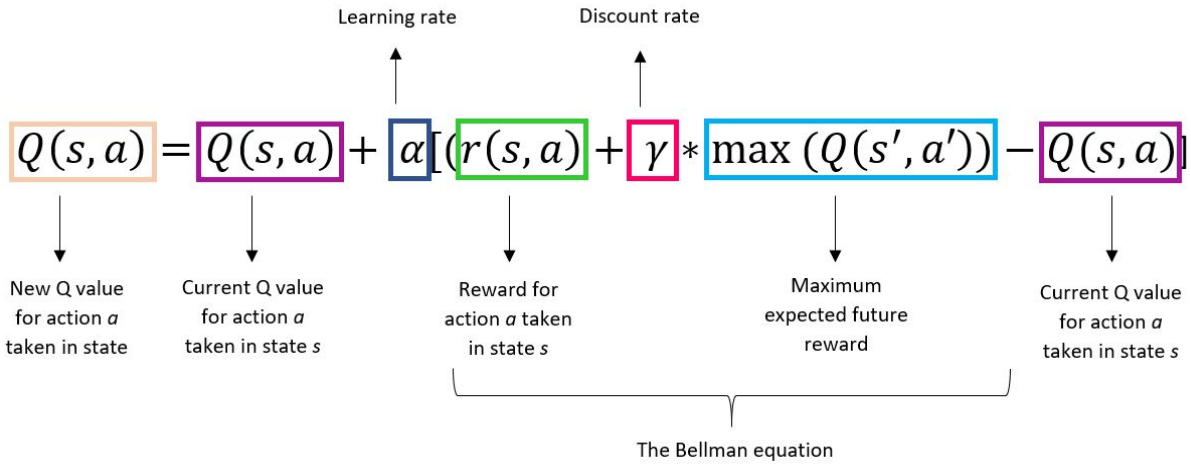
In short: **The value of an action = Immediate reward + maximum possible reward from next state * discount factor**

Q-LEARNING

Quality learning, or 'Q-learning', makes use of both the Markov decision process and the Bellman equation as part of an algorithm to determine which action to take in each state. Using these two aspects means that we know that a future state only depends on the current state and action (with reward) and that we can estimate which action to take by determining its overall Q-value.

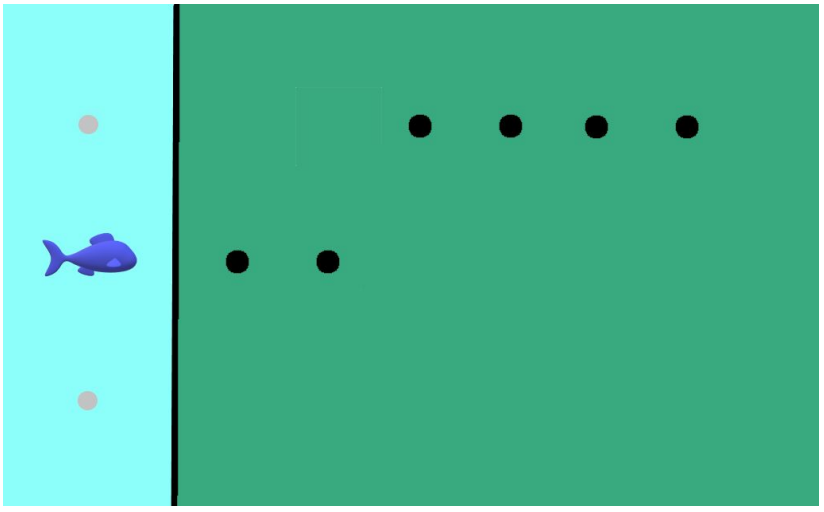
A Q-table is used to keep track of how much reward taking an action from a state could achieve. In its simplest implementation, a Q-table has a row for each state and a column for each action. Through training, the values in this table are updated **to reflect the value estimation for each action from each state**, based on both the current reward and the expected future reward from the next state. Thus, the Q-table guides the agent as to which is the optimal action to take from a state by providing a value for how good each action is.

Mathematically, the Q-learning algorithm updates the Q-table using this equation:



Let's have a look at a simple example of this.

Imagine a simple game where food moves towards a fish and using three actions (remain still, go up, do down) the aim of the game is to make sure it eats as much food as possible. The fish can occupy one of three positions (states): the one it is currently shown below, as well as the two grey spots. The top spot is state 0, the middle is state 1 and the bottom is state 2.



The Q-table for this example would initially look something like this, where we have each of the three states as rows and the possible actions as columns.

States | Actions	**Remain still**	**Go up**	**Go down**
0	0	0	0
1	0	0	0

2	0	0	0

The rules of the game are that if the fish eats a piece of food, it gets the reward +10 and if it doesn't eat food on a move it gets a punishment of -20. In this example, we will use the learning rate of 0.1 and a discount factor of 0.75.

The first time this game is played, the agent doesn't know anything about which actions are good, so the agent randomly selects to remain still on its first move. The first piece of food right in front of the fish moves over the line and the fish eats it. The agent gets 10 points, so we update the Q table to reflect this.

Q(1,still) = 0 + 0.1[10 + 0.75 * max(0,0,0) -0] = 1

Note: the max Q-value is the maximum value from the new state's row. Ie: which action looks most promising given past experience in the state we are moving to (even if that means that we are 'moving' to the same state)?

	Remain still	**Go up**	**Go down**
0	0	0	0
1	1	0	0
2	0	0	0

The agent then considers that it is again in state 1. It considers the Q-table and sees that it has had success when selecting to stay still when in this state before, so again it chooses to stay still, eats the next piece of food and is given a reward of +10. The Q-table is then updated again.

Q(1,still) = 1 + 0.1[10 + 0.75 * max(1,0,0) - 1] = 1.98

	Remain still	**Go up**	**Go down**
0	0	0	0
1	1.98	0	0
2	0	0	0

On its third try, the agent considers the Q-table again for state 1, since it hasn't moved yet, sees that staying still has really worked for it so far so it stays still again — but there is no food so it gets a punishment of -20.

Q(1,still) = 1.98 + 0.1[-20 + 0.75 * max(1.98,0,0) - 1.98] = 1.43

	Remain still	Go up	Go down
0	0	0	0
1	-0.07	0	0
2	0	0	0

What this means is that the next time the agent has to choose which way to move, it will randomly select between the actions go up or go down — as the option to remain stationary is not attractive anymore.

Over time the agent will learn that the most optimal strategy for this game is to eat the immediate food and then move up. It will learn this over several iterations of the game. The fundamental aspect of Q-learning is the adjustment of the Q-table based on the reward obtained and that expected next. The learning rate and discount factor simply tweak this behaviour; although these need to be selected carefully. The learning rate needs to be big enough that learning is done, but not so big that we adjust the values by too much. On the other hand, the discount function lets us define how much we should consider future rewards when choosing our current action.

If you're interested, an excellent blog post about Q-learning can be found **here**.

THE WIDER WORLD OF REINFORCEMENT LEARNING

Reinforcement learning has been used in many exciting applications, including beating the world champion Go player (Go is considered the hardest strategy board game in the world). A documentary was made on this called "AlphaGo" and is available on YouTube.

Instructions

The exercises in these tasks require users to run Jupyter Notebooks through the Anaconda environment. Please make sure that you download the Anaconda Distribution from **https://www.anaconda.com/distribution/**.

Compulsory Task

Launch Jupyter Notebook via your Anaconda environment and upload the file named **reinforcement_learning_task.ipynb** from your task folder and follow the instructions.

Completed the task(s)?

Ask your mentor to review your work!

Review work

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask your mentor for alternative instructions.

Rate us

Share your thoughts

Hyperion strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.