Reinforcement Learning

Rosie Martinez

Data Science Immersive

Learning Objectives

By the end of this lecture you will:

- Define what reinforcement learning is
- Describe the main components of reinforcement learning
- Identify current applications of reinforcement learning

What is RL?

Reinforcement Learning (RL) can be considered a third type of Machine Learning:

- Supervised Learning: from features predict a target
- Unsupervised Learning: find structure in the data
- Reinforcement Learning: Let an **agent** interact with an **environment** and use feedback from the environment to train its behavior
    - Some might call this semi-supervised

What is RL?

RL hasn’t always been assisted by deep learning, but a fundamental challenge of RL is how to represent the data. Deep learning is very good at engineering features to represent data.

A more exact definition of RL:

In RL an **agent** is placed in an **environment** where it receives **rewards** for its **actions** depending what **state** the **agent** and **environment** are in

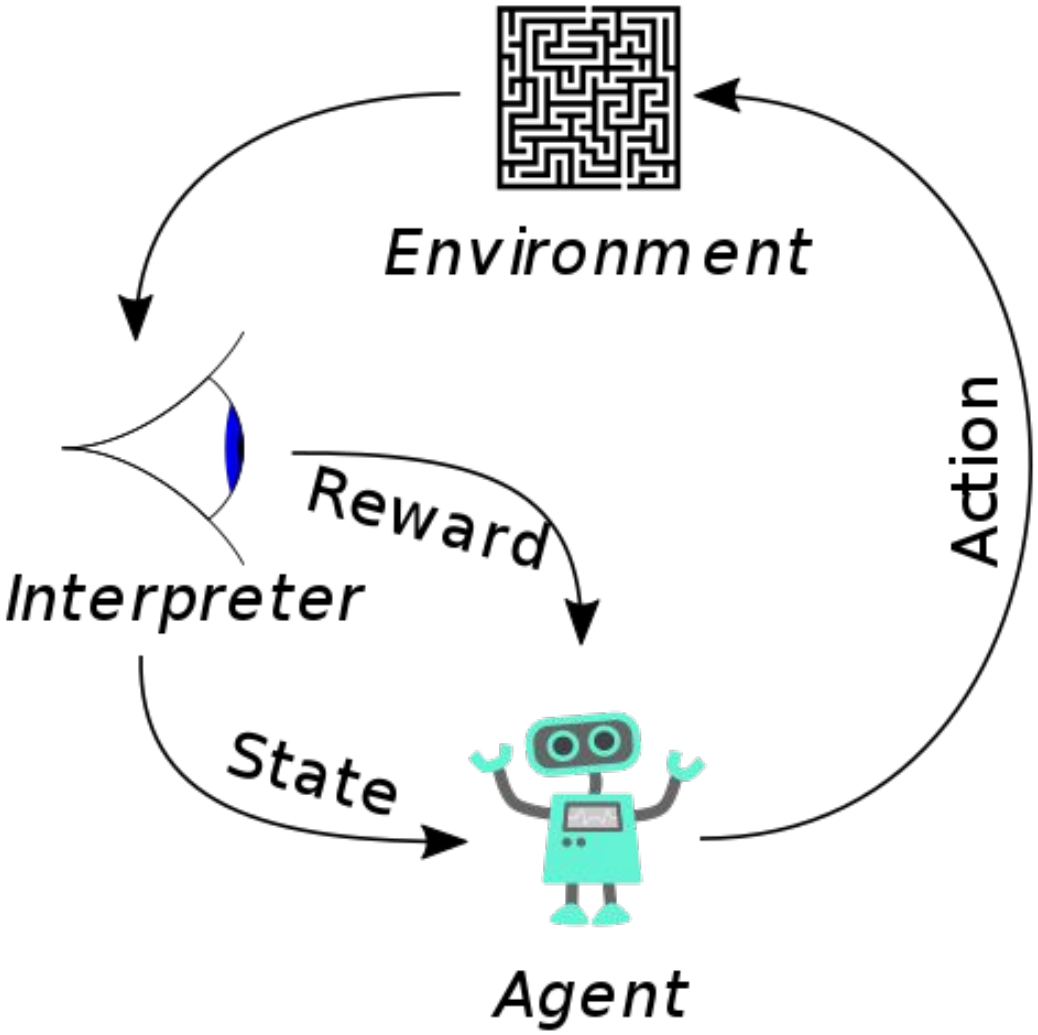
Components of RL

The key idea behind RL:

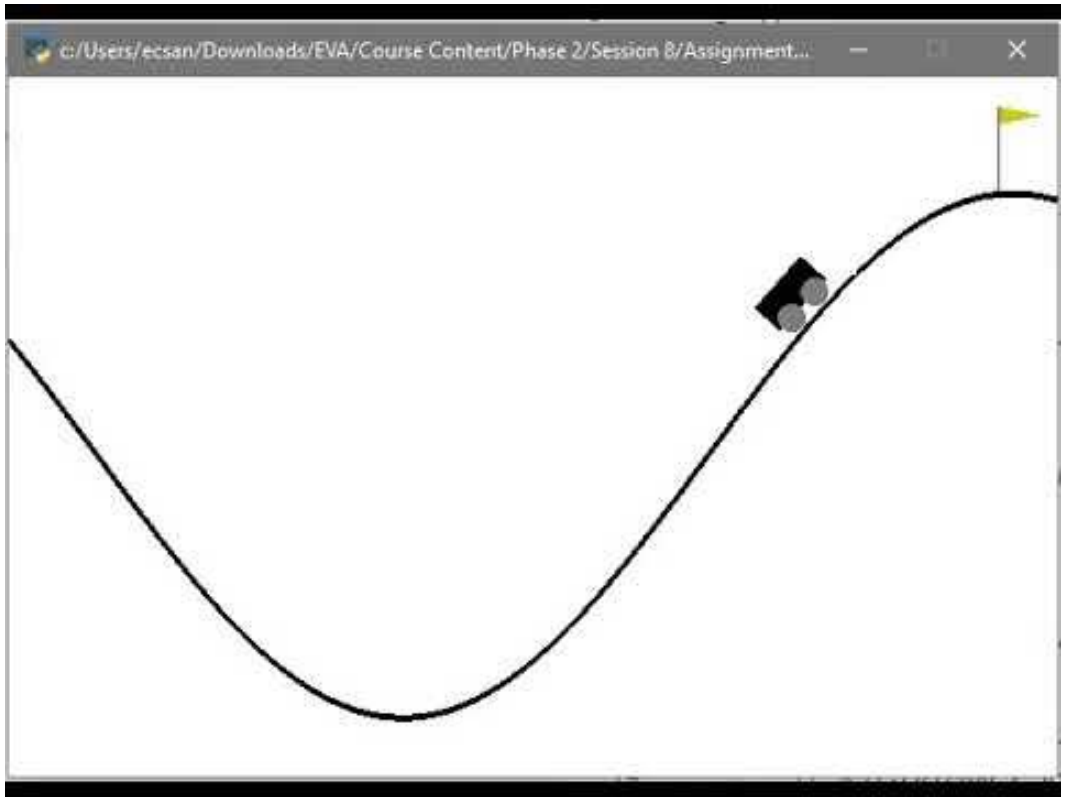
- We have an environment which represents the outside world to the agent
- We have an agent that takes actions, receives observations from the interpreter that consists of a reward for its action and information of its new state
- That reward informs the agent how good or bad was the taken action
- The interpreter tells it what its next state in the environment

Each iteration is called an episode, this might be after some time, *t* (like an epoch)

[Source]



Let’s walk through an example… Mountain Car



Source

Components of RL: Agent

The **Agent** is received the current state and reward.

The agent tries to figure out the best action to take or the optimal way to behave in the environment in order to carry out its task in the best way

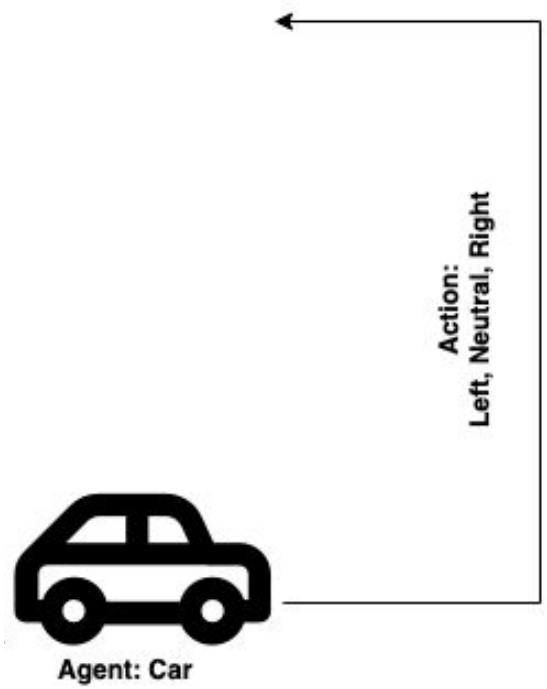
In our case, the agent is the Car

Agent: Car

Components of RL: Action

The **Action** is what the agent can do within the given environment.

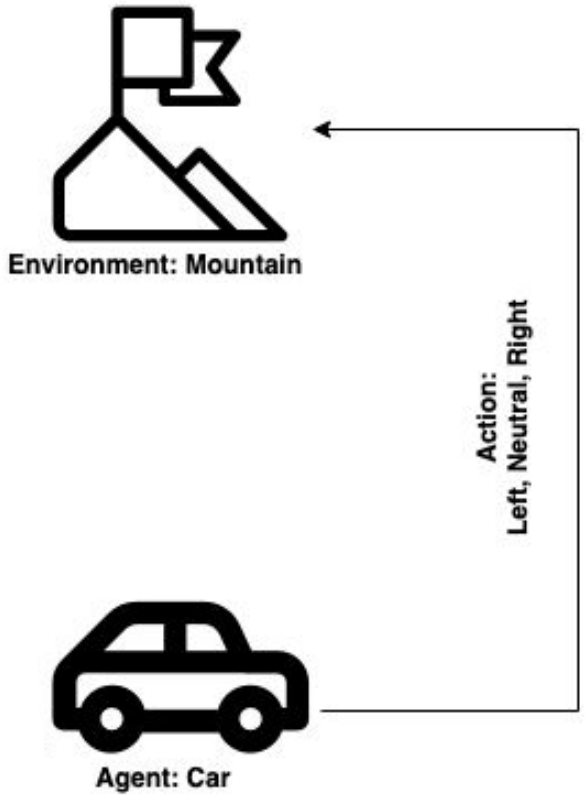
In our case, our agent can take actions Left, Neutral, or Right.



Components of RL: Environment

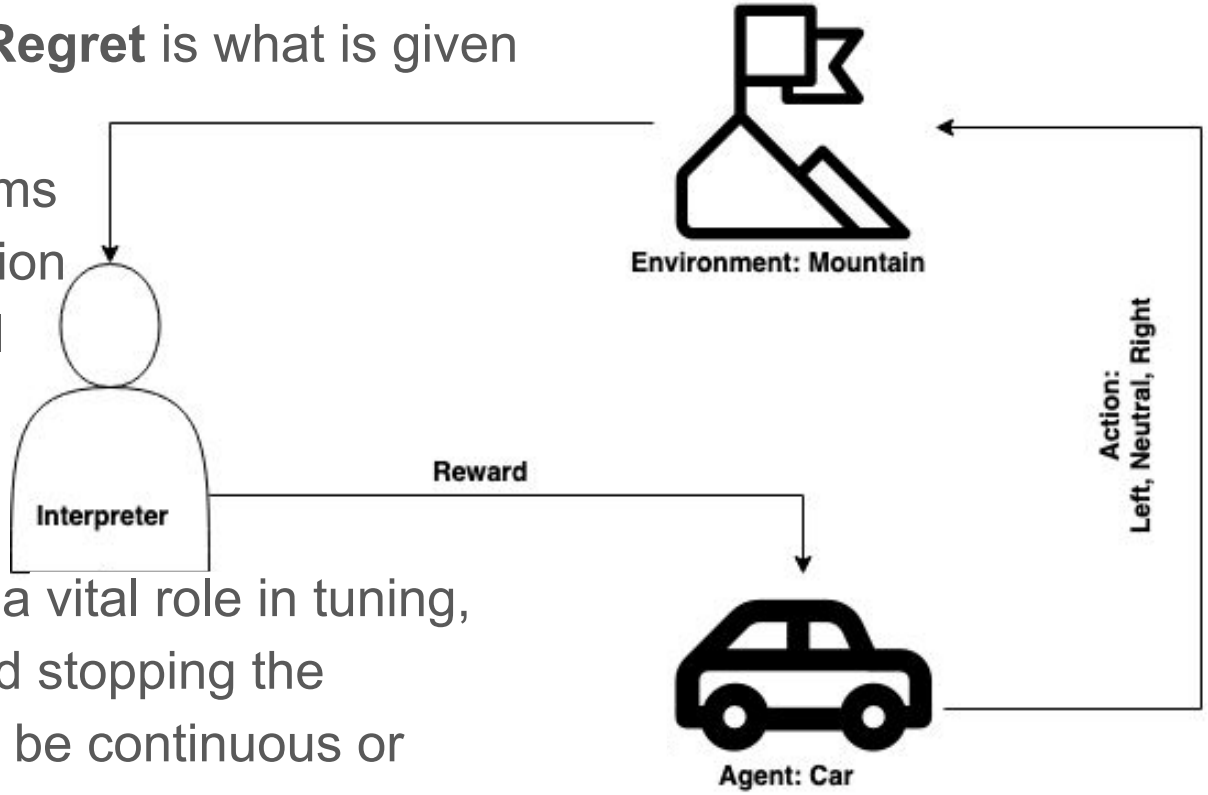
The **Environment** is area in which the agent takes its action, has current states, and receives rewards. The agent interacts with the environment in discrete time steps

In our case, our environment is the Mountain



Components of RL: Reward/Regret

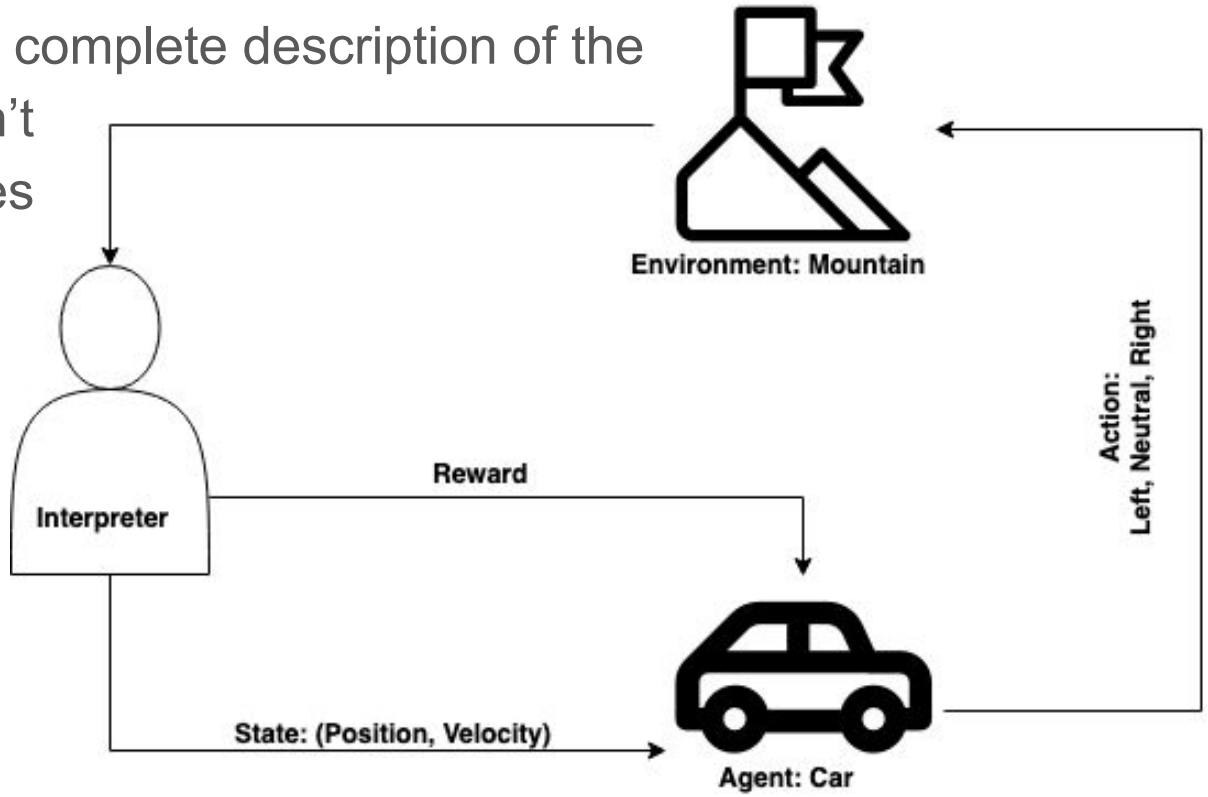
The **Reward/Regret** is what is given to the agent when it performs an optimal action or non-optimal action.



Reward plays a vital role in tuning, optimizing, and stopping the algorithm, can be continuous or discrete, positive or negative

Components of RL: States

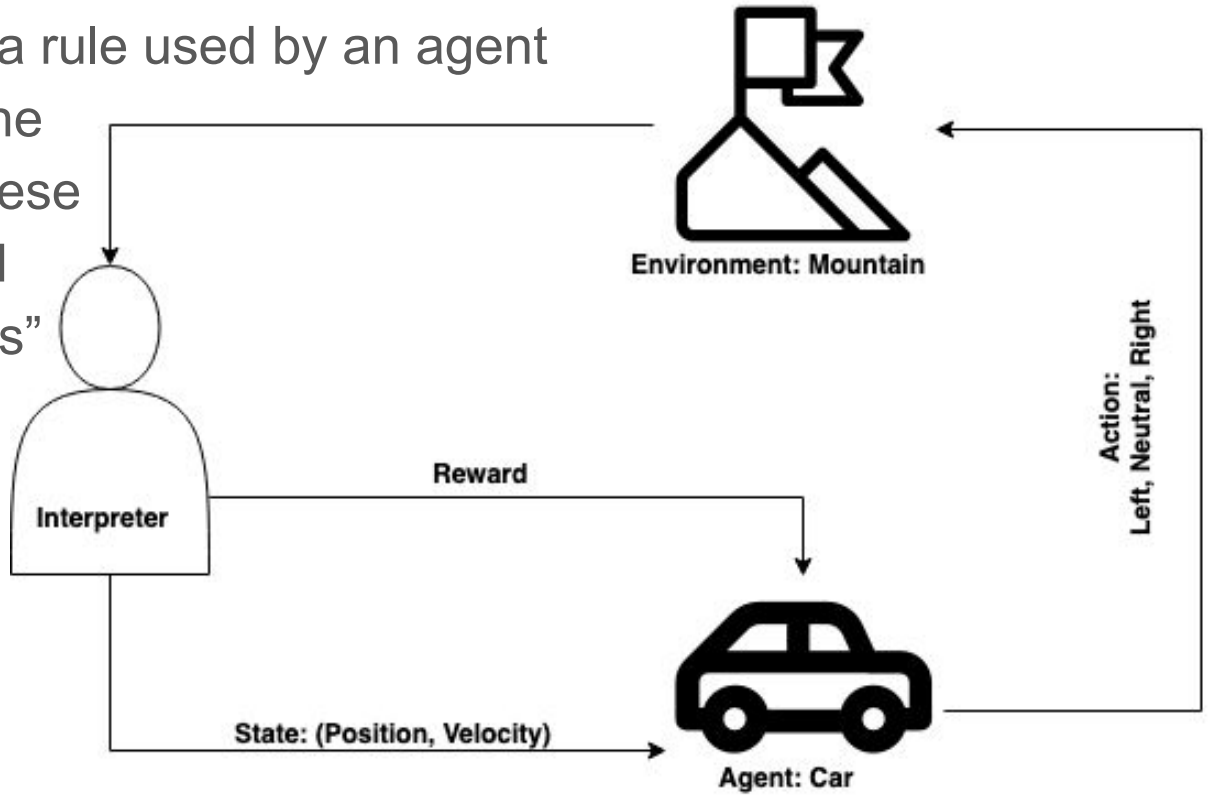
The **State** is a complete description of the world, it doesn't hide any pieces of information that is present in the world. It can be a position, a constant, or a dynamic.



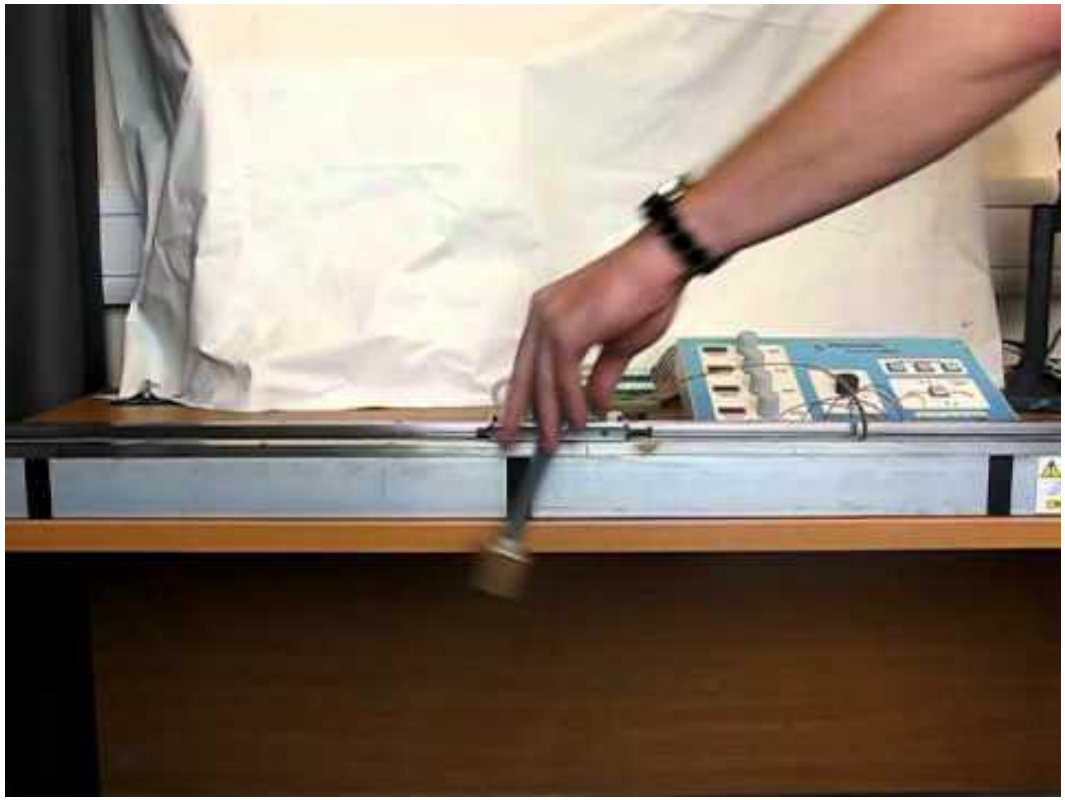
In our case, our states are position and velocity

Components of RL: Policy

The **Policy** is a rule used by an agent for choosing the next action, these are also called “Agent’s Brains”



Breakout: Cart Pole Example



This is a classic example of the control theory pole balance problem

Breakout: In the cart pole video …

- What is the **agent**?
- What is the **environment**?
- What are the **actions** the **agent** can take?
- What is the **reward**?
- What is the **state**?
- Would you describe this as a deterministic approach or trial and error?

How can we approach RL?

There are several ways we can approach a RL model, however there are three main ways:

1. The Naive Model: Taking an educated guess, rule of thumb
2. Q-Learning: Accounting for a Delayed Reward
3. Deep Learning Models

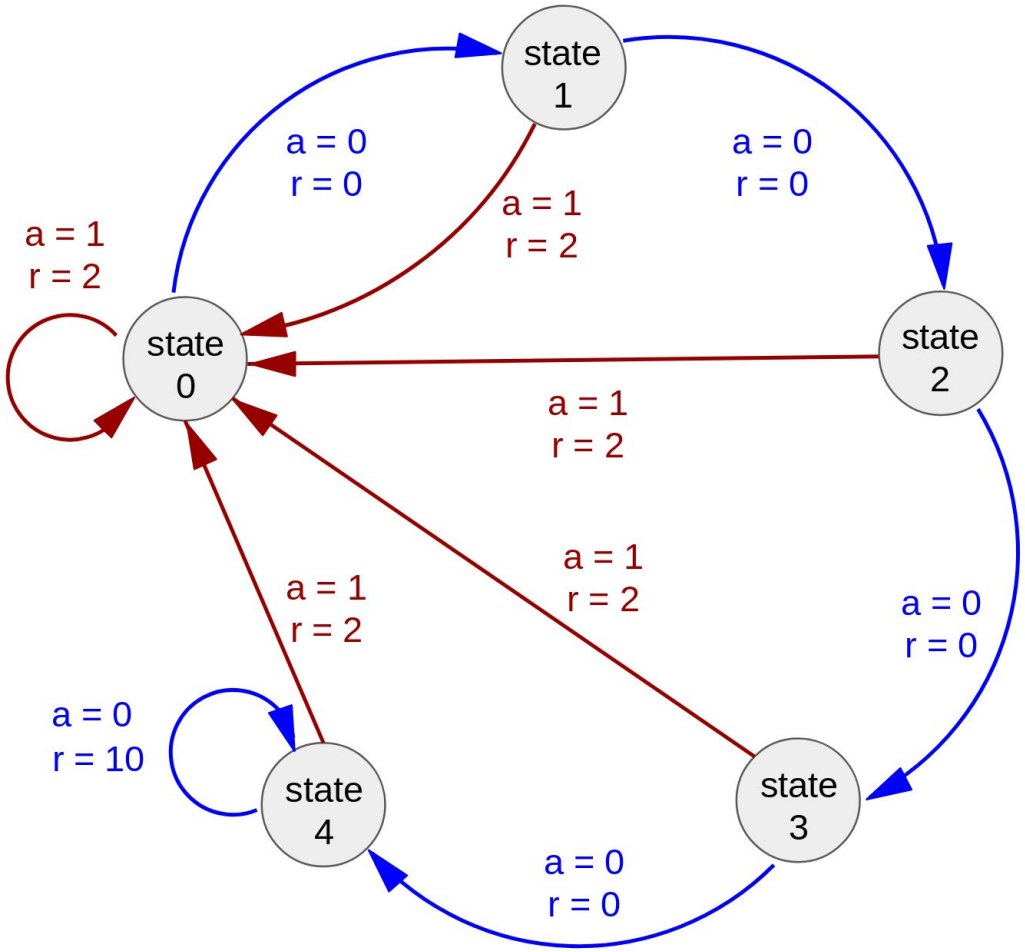
The example we will be coding: NChain-v0 from OpenAI Gym

NChain OpenAI Gym Environment
5 states
2 actions possible in each state

Actions:
action 0 (a = 0) proceed clockwise from present state
action 1 (a = 1) return to state 0

Rewards:
0 when a = 0 (unless in state 4)
2 when a = 1

Stochastic behavior:
Sometimes the action taken will be opposite what was requested, and the award associated with the actual action will be applied.
e.g. a = 0 requested, but
a = 1 taken, r = 2



The Naive Model

We need to find a **policy** π which maps **states, actions,** and **rewards** in such a way that **the cumulative award is maximized** in the episode

Imagine a **reward table:**

state	action 0	action 1
0	$\sum_{step=1}^{N} r_{a0}$	$\sum_{step=1}^{N} r_{a1}$
1	$\sum_{step=1}^{N} r_{a0}$	$\sum_{step=1}^{N} r_{a1}$
2	$\sum_{step=1}^{N} r_{a0}$	$\sum_{step=1}^{N} r_{a1}$
3	$\sum_{step=1}^{N} r_{a0}$	$\sum_{step=1}^{N} r_{a1}$
4	$\sum_{step=1}^{N} r_{a0}$	$\sum_{step=1}^{N} r_{a1}$

Where a simulation fills in the values in this table

The Naive Model

Let’s jump to a jupyter notebook to see this in action…

Based on our example we have problems with this approach:

- We aren’t exploring both actions at each state
- We know that the best solution is to take action 0 all the way to state 4 and then continue to take action - there, but the table above isn’t showing that (most the time is spend in state 0)
- This approach only sums immediate rewards for an action taken in a given state. It has not mechanism for accounting for a delayed reward (picking action 0 with no reward until state 4).

Q-Learning

The idea of propagating possible reward from the best possible actions in future states is a core component of what is called Q-learning.

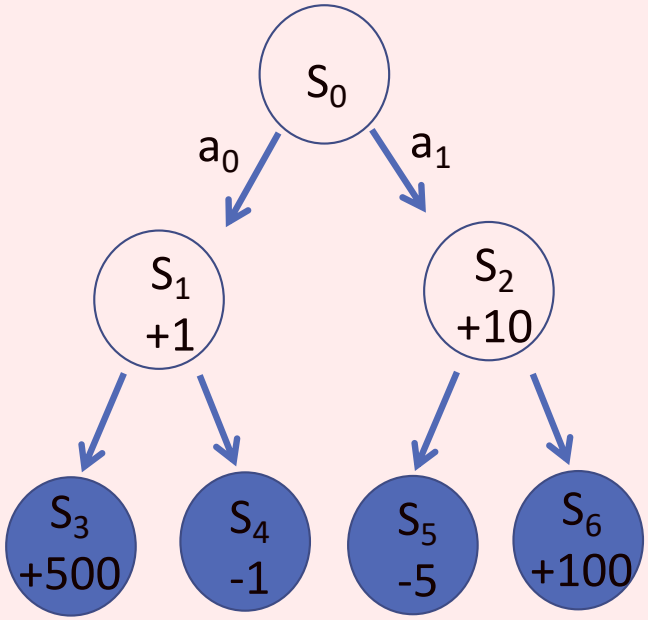
The Q-learning update rule (Bellman Equation):

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

where r_t is the reward observed for the current state s_t, and α is the learning rate ($0 < \alpha \leq 1$).

And γ is a discount factor ($0 < \gamma \leq 1$) associated with how much weight to future rewards. The $\max Q(s_{t+1}, a_{t+1})$ is the maximum Q value obtained from any action in that state. (source)

The Game



Episodic Game:

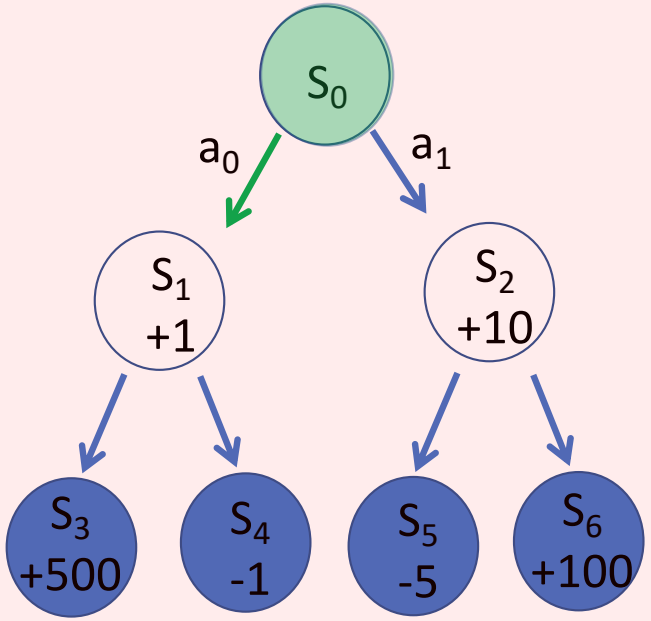
- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

	Left	Right
S_0	0	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

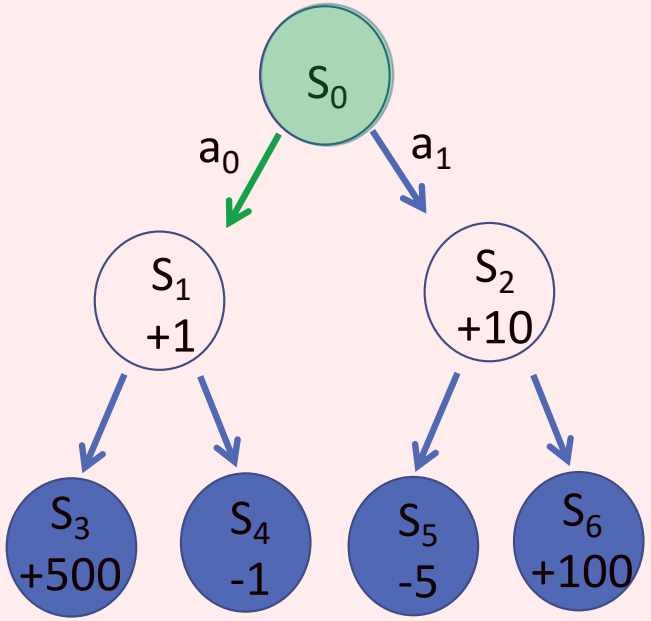
	Left	Right
S_0	0	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

Given we are in State 0, select a random action, Left, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_0, a_0) = 0.5 * 0 + 0.5 * (1 + 0) = 0.5$$

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

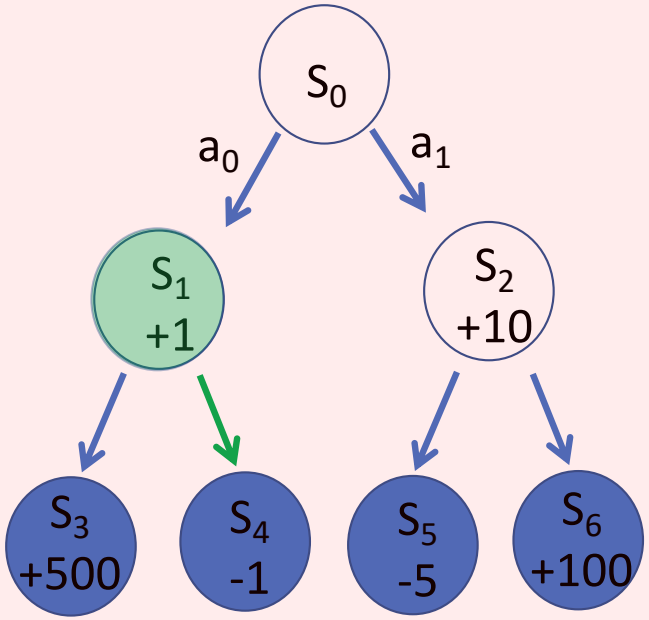
	Left	Right
S_0	0.5	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

Given we are in State 0, select a random action, Left, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_0, a_0) = 0.5 * 0 + 0.5 * (1 + 0) = 0.5$$

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

	Left (0)	Right (1)
S_0	0.5	0
S_1	0	0
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

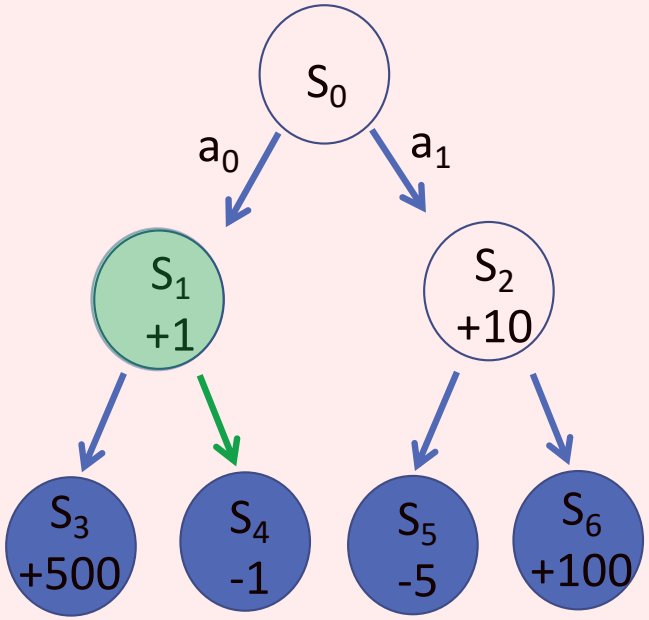
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal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

Given we are in State 1, select a Random action, right, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_1, a_1) = 0.5 * 0 + 0.5 * (-1 + 1 * 0) = -0.5$$

The first episode is complete!

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

	Left (0)	Right (1)
S_0	0.5	0
S_1	0	-0.5
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

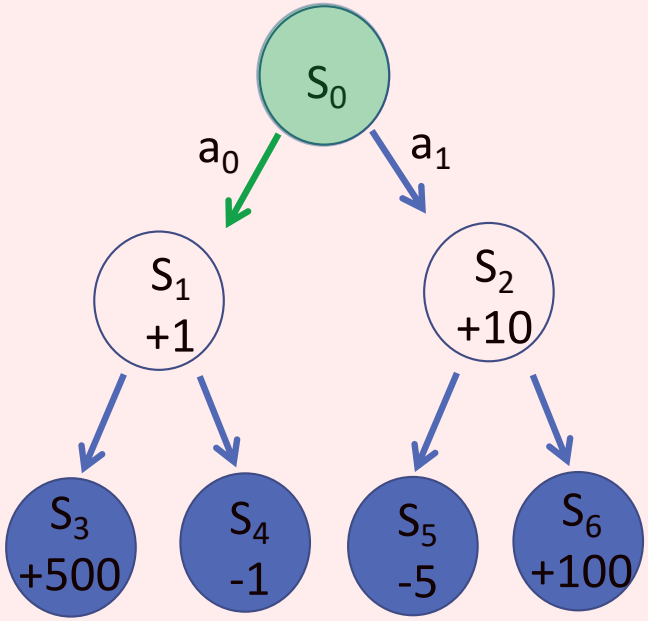
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal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

Given we are in State 1, select a Random action, right, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_1, a_1) = 0.5 * 0 + 0.5 * (-1 + 1 * 0) = -0.5$$

The first episode is complete!

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

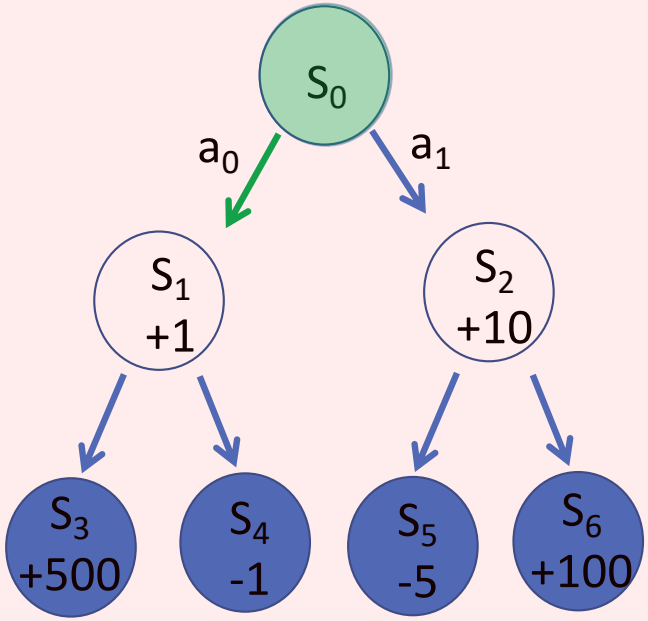
	Left (0)	Right (1)
S_0	0.5	0
S_1	0	-0.5
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

Given we are in State 0, select the action with the highest q-value, left, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_1, a_1) = 0.5 * 0.5 + 0.5 * (1 + 1 * 0) = 0.25 + 0.5 = 0.75$$

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

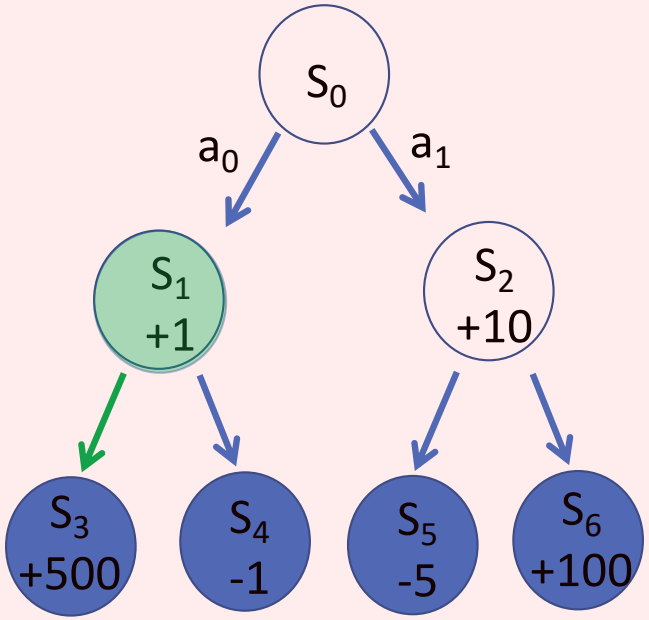
	Left (0)	Right (1)
S_0	0.75	0
S_1	0	-0.5
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

Given we are in State 0, select the action with the highest q-value, left, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_1, a_1) = 0.5 * 0.5 + 0.5 * (1 + 1 * 0) = 0.25 + 0.5 = 0.75$$

The Game



Episodic Game:

- **States are circles below**
- **Environment is tree below**
- **Rewards are defined on each circle**
- **Game terminates when you reach a solid blue circle**
- **Goal is to maximize the returns**

Q-table

	Left (0)	Right (1)
S_0	0.75	0
S_1	250.5	-0.5
S_2	0	0
S_3	0	0
S_4	0	0
S_4	0	0
S_6	0	0

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

Given we are in State 1, select the action with the highest q-value, left, and update the Q-table. Let alpha =0.5 and gamma = 1.

$$Q_{new}(s_1, a_0) = 0.5 * 0 + 0.5 * (500 + 1 * 0) = 250.5$$

Q-Learning

Let’s jump to a jupyter notebook to see this in action…

If everything worked out, the Q values under action 0 should be larger than action 1 for all states. Therefore, the best **policy** to **maximize our cumulative reward** is to choose action 0 in every state.

This is a simple environment, so a simulation like this can be performed to find the Q values and determine the best policy. But most likely the environment is more complicated that this, with more states and actions possible at each state.

However, neural networks can be trained to predict Q values for each action in a given state in more complicated problems. Deep learning to the rescue!

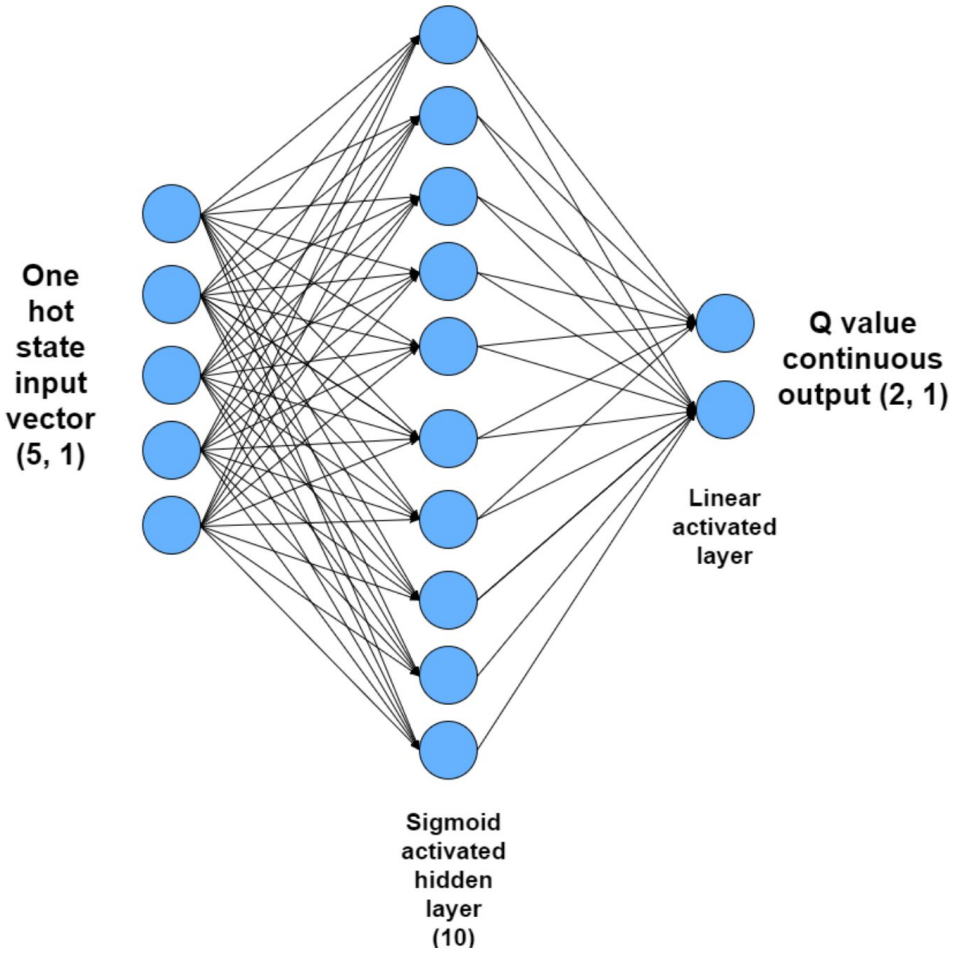
Deep Learning

To develop a neural network which can perform Q learning, the input needs to be the current state (plus potentially some other information about the environment) and it needs to output the relevant Q values for each action in that state. The Q values which are output should approach, as training progresses, the values produced in the Q learning updating rule. Therefore, the loss or cost function for the neural network should be:

$$\text{loss} = \underbrace{(r + \gamma \max_{a'} Q'(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}})^2$$

Deep Learning

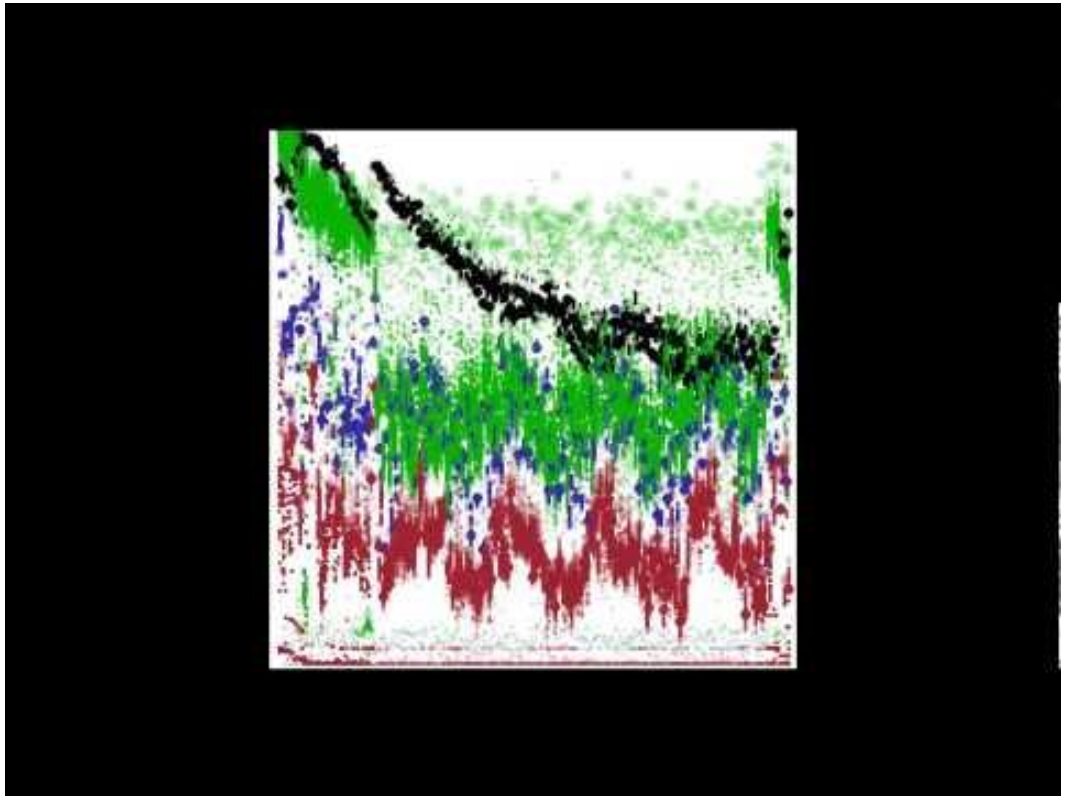
We will use this architecture:



Current Applications of RL

- Traffic Light Control
- Robotics
- Web System Configuration
- Bidding and Advertising

Tetris Example: What did the agent learn?



Other RL Environments

Other Environments are available in OpenAI Gym: environments

Solutions to some of these environments can be found on the OpenAI Leaderboard

You can also make your own environment: Guide

A past student's RL Capstone Project:

Kyle Schutter: Teaching a NN to play Connect4

A past case study in the DSI from a few classmates and myself:

https://github.com/rosiemin/Reinforcement-Learning-Case-Study