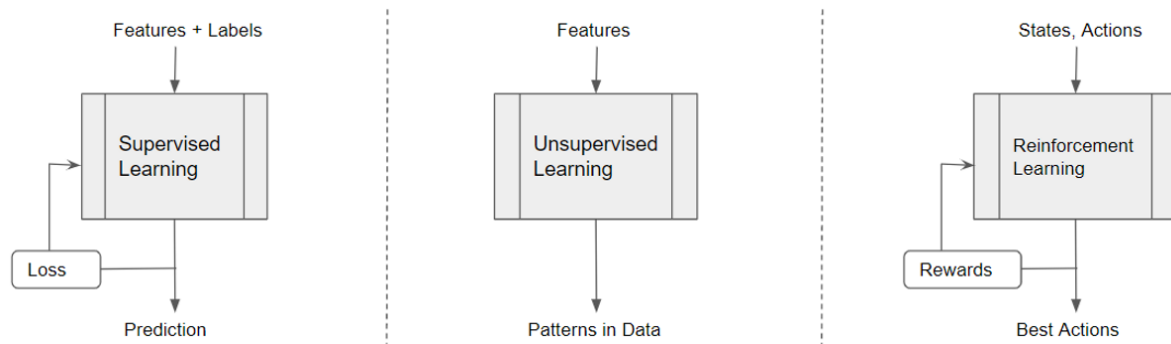


Reinforcement Learning

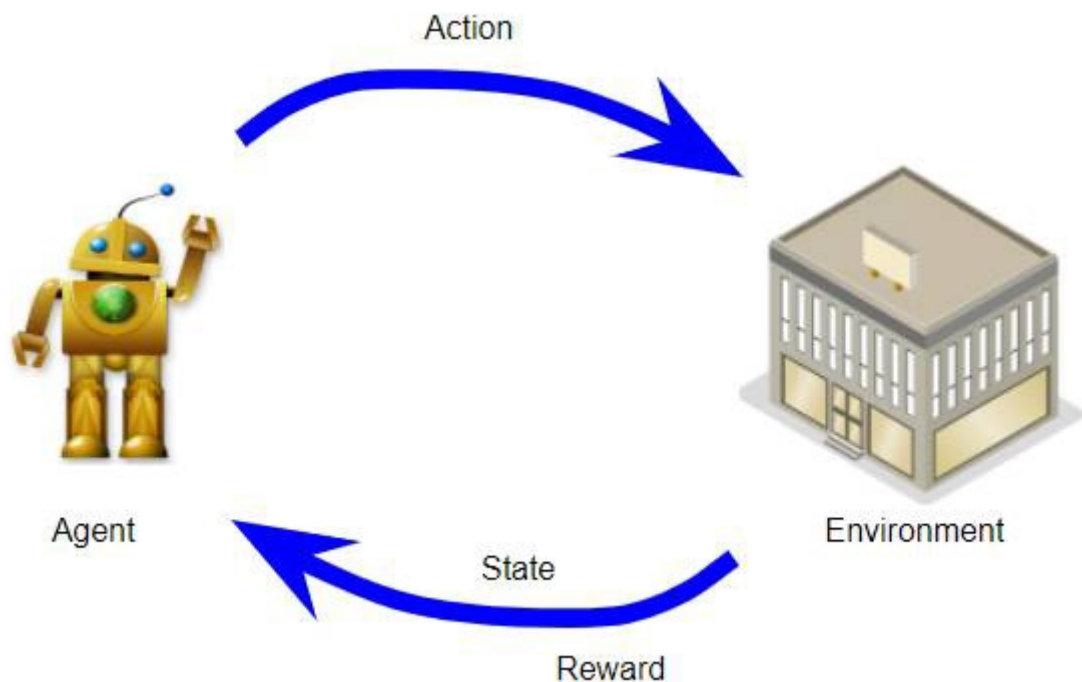
Reinforcement learning is a subset of machine learning where an **agent** is trained to achieve a goal in an **environment**. The agent *actively senses the **state** of the environment, performs an **action** based on the environment, maximizes the positive **reward** obtainable from the environment (the goal) and learns a **policy** (a state-to-action mapping) that performs best in the environment.*

Reinforcement learning treads the line between two other popular machine learning methods, supervised and unsupervised learning.



In supervised learning, the model training is supervised through a set of training data where each example in the data set is labeled to represent some desired output. The purpose of such a trained model is to generalize to the dataset or extrapolate its responses so that it acts in accordance to the underlying process that generated the dataset. Unsupervised learning, is finding the hidden structure in an unlabeled dataset. Both these kinds of learning, though important, have some drawbacks when it comes to complex environments where the agent needs to learn from its own experiences by probing an uncertain environment. Successful supervised learning models require large amounts of labeled data, reinforcement learning algorithms, on the other-hand require learn from sparse reward signals in noisy and delayed environment. Another drawback of other deep learning techniques is that they assume that all training data samples are independent from each other, where as the environment that a reinforcement learning agent has to operate is in has states that are highly correlated. Furthermore, as the reinforcement learning environment is uncertain and the underlying state-action distribution is constantly changing training supervised learning model is highly unstable.

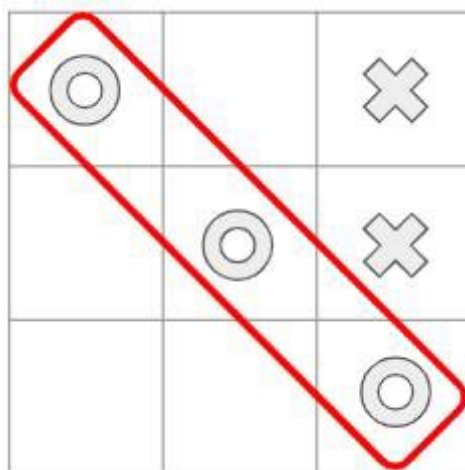
An agent in reinforcement learning environment can be described through a Markov Decision Process (MDP). Simply, MDPs simply incorporate three aspects in their decision process; sensation (the state of the environment), action (action taken by the agent in the current state) and goal (the reward obtained by performing the action).



Since an agent in a reinforcement learning environment has to actively probe the environment for reward there are two challenges that arise, how much to **explore** the environment and how much to **exploit** the environment. An agent needs to maximize the reward it accumulates in the environment, its best shot would be to exploit the environment's reward system using past experiences and choose the best action according to it, but to choose such actions it first has to explore the environment through trial-and-error to see what works and what does not. The trade-off between exploration and exploitation is still an active research area.

Operations of a Markov Decision Process in a Reinforcement Learning environment.

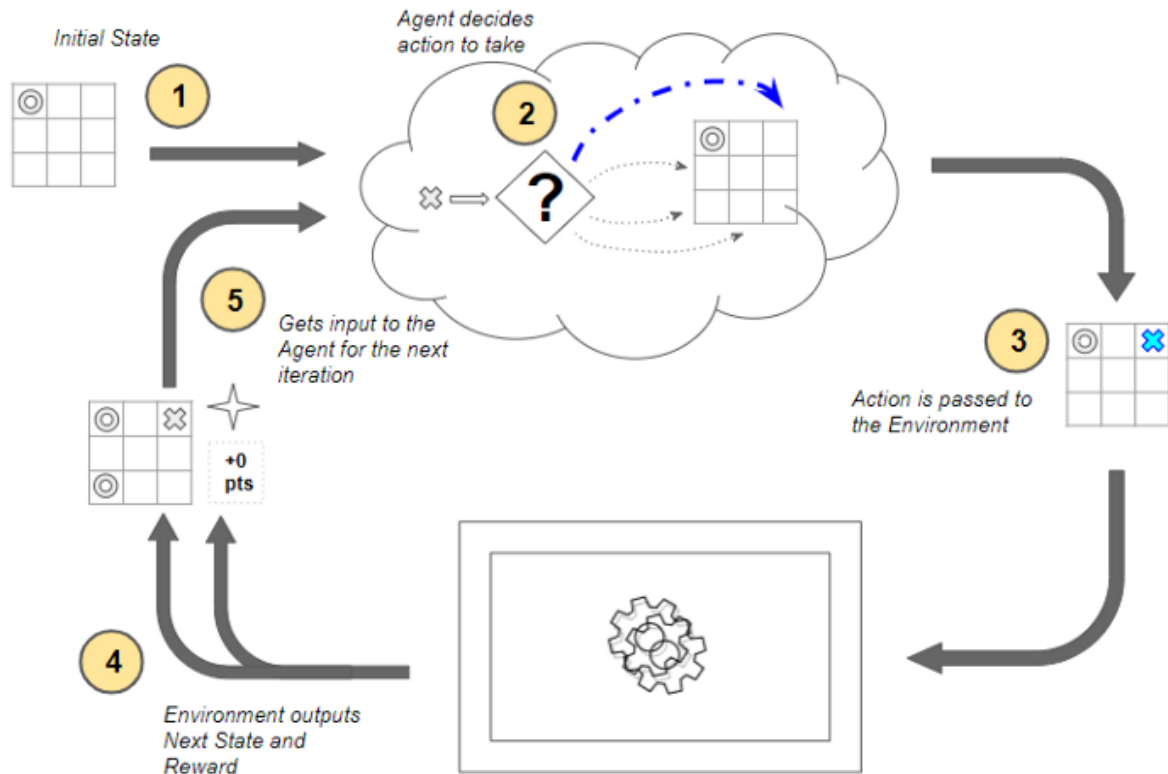
Now that we've defined an overview of a reinforcement learning environment and agent. Let's describe the decision processes that constitute the entire environment in a systematic way, using the Tic-Tac-Toe game.



In Tic-Tac-Toe one player places X's on the 3x3 grid while the other places O's, whoever is able to get first place three of their symbols in a straight line wins. This can be defined through a Markov decision process (MDP) as follows:

- The agent plays against the environment, the opponent
- The state is the current placement of all the symbols on the grid
- There are 9 possible actions when the game first starts and are reduced as the game proceeds.
- If the agent wins it receives a positive reward of +1, if it loses it receives a reward of -1. Each of the intermediate moves simply provide no reward, i.e 0.

This entire MDP process can be visualized as follows:

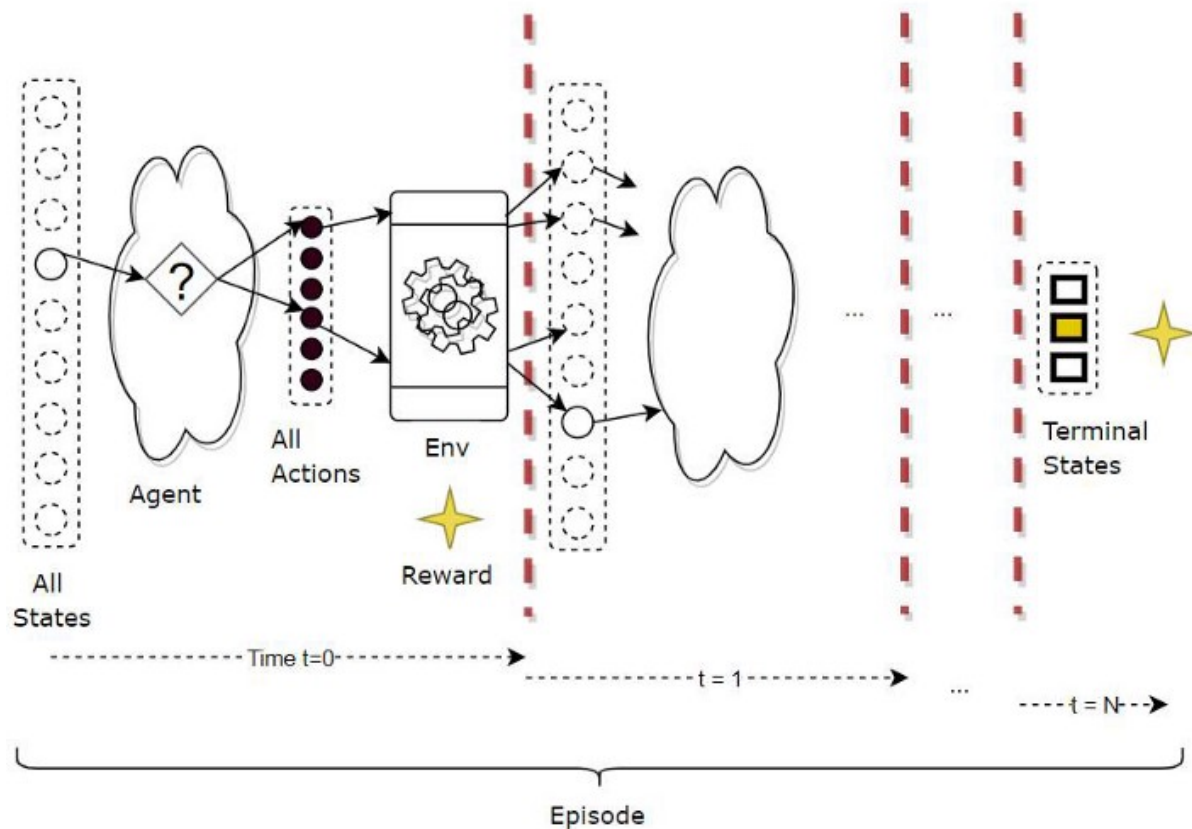


Here's what's happening at each time-step:

1. The environment's current state is the input to an agent.
2. Based on the input state the agent takes an action.
3. The action causes a reaction in the environment
4. The environment using the new state performs an action and also returns a reward to the agent.
5. The agent then using the new state and reward as feedback performs another action.

This entire process is just one time-step. Throughout the process the agent's main task is to maximize the reward it receives from the environment.

A MDP in a reinforcement learning environment can be seen as interacting with the environment at each time-step and trying to perform the best action according to some rubric.



The rubric according to which the actions should take place is what reinforcement learning is.

So, how does the agent pick an action?

For this we have to define three concepts:

- Return
- Policy
- Value

Return: Return is the cumulative reward that the agent gathers as it interacts with the environment.

Policy: Is the strategy according to which the agent defines the mapping between states and actions that results in the maximum reward across a series of time-steps called an episode.

Value: Is the expected return that an agent can receive across multiple episodes.

The relationship between reward, return, and value is as follows:

- *Reward* is immediate following an action.
- *Return* is the total of all the rewards.
- *Value* is the average return over multiple episodes.

How to define Return?

Return is not a simple addition of rewards across time-steps, that could result in explosion of rewards. Instead return is defined as the discounted reward over time. In simple terms, when we calculate return we add the immediate reward plus the rewards that the agent will receive in the subsequent step exponentially weighted by some factor less than 1, called γ (gamma).

$$Return_0 = Reward_0 + \gamma Reward_1 + \gamma^2 Reward_2 + \gamma^3 Reward_3 + \dots$$

The above equation is much easier to decipher when we look at this from the terminal (ending) reward backwards. For example the episode of a reinforcement learning agent ends at time-step $t = 3$ and the reward at that time-step is $Reward_3$, since it's the terminal step the return (cumulative reward) is simply $Return_3 = Reward_3$, from this we can trace backward and arrive at the above equation applying the weight called the **discounting factor**, γ , at each time-step:

$$Return_3 = Reward_3$$

$$Return_2 = Reward_2 + \gamma(Return_3) = Reward_2 + \gamma Reward_3$$

$$Return_1 = Reward_1 + \gamma(Return_2) = Reward_1 + \gamma Reward_2 + \gamma^2 Reward_3$$

$$Return_0 = Reward_0 + \gamma(Return_1) = Reward_0 + \gamma Reward_1 + \gamma^2 Reward_2 + \gamma^3 Reward_3$$

This shows us that:

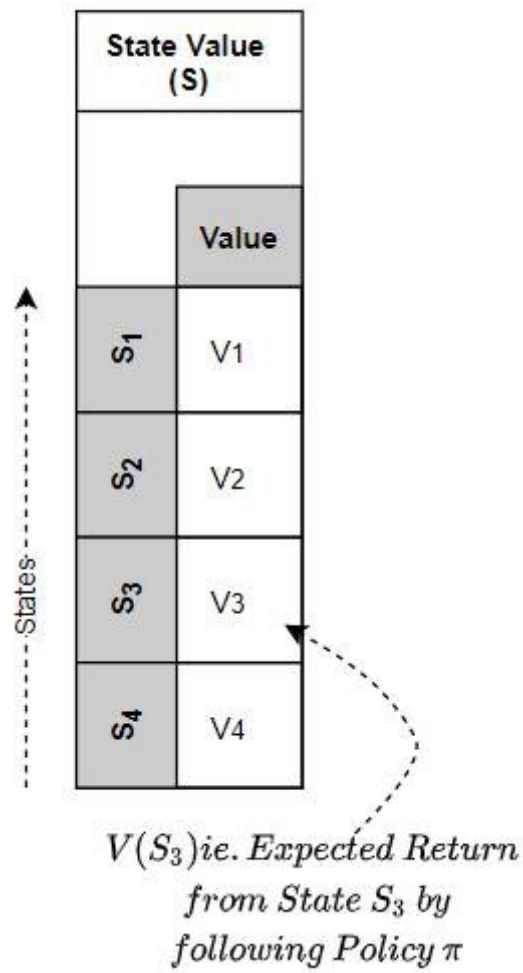
- Discounting the rewards exponentially prevents blowing up of Return as the episode progresses.
- It encourages immediate reward as they are the least discounted.
- The ultimate goal should be maximizing reward across the episode.

How do we define Value?

As stated previously value tells us the average return over multiple episodes when we follow a policy. Value is important as it helps us differentiate between policies as we try to find the optimal policy for the environment.

Value can be seen as a lookup table that maps state, or state-action pairs to a value that can be obtained following a policy.

- State Value: average return from a given state, when using policy π ,



State Value (S)	
	Value
s_1	V1
s_2	V2
s_3	V3
s_4	V4

$V(s_3)$ ie. Expected Return from State s_3 by following Policy π

- State-Action Value (aka Q-Value): average return when taking a given action from a given state following policy π .

State-Action Value (S x a)				
States ↑	Actions →			
	a ₁	a ₂	a ₃	
	s ₁	Q ₁₁	Q ₁₂	Q ₁₃
	s ₂	Q ₂₁	Q ₂₂	Q ₂₃
	s ₃	Q ₃₁	Q ₃₂	Q ₃₃
s ₄	Q ₄₁	Q ₄₂	Q ₄₃	

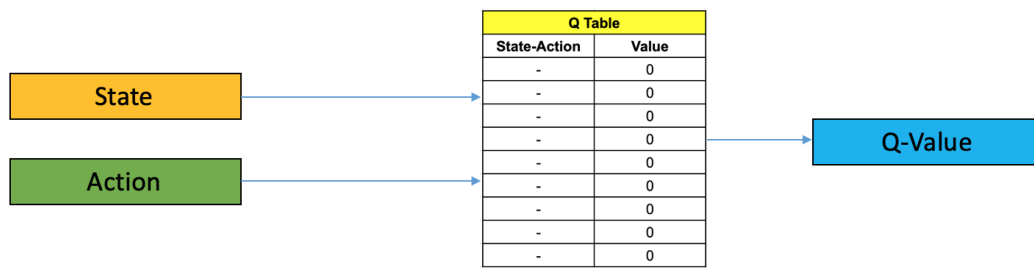
$Q(s_3, a_3)$ i.e. Expected Return by taking Action a_3 from State s_3 and following Policy π after that

In order to create an intelligent agent in the environment we need to figure out the an algorithm to create the optimal State-Action Value, Q-value, lookup table, that will basically give us the best action to perform at each time-step which will give the maximum Return.

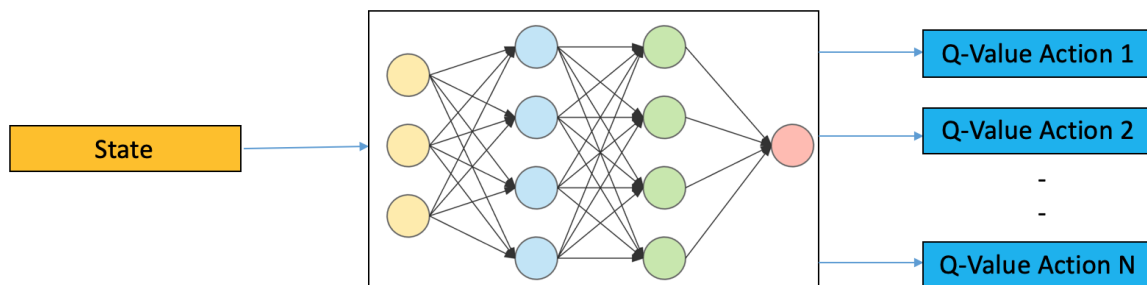
How to find the Q-value?

Since Q-values are a mapping of actions in response the states we can create a neural network to act as an approximator to the Q-value table. Neural networks are universal function approximators , so given an neural network with sufficient capacity (architecture, depth, width etc.) and sufficient amount of data we can approximate the Q-value table that is optimal for the agent's environment.

This is called Deep-Q Learning. Its succinctly described by the following figure:



Q Learning



Deep Q Learning

The DQN algorithm

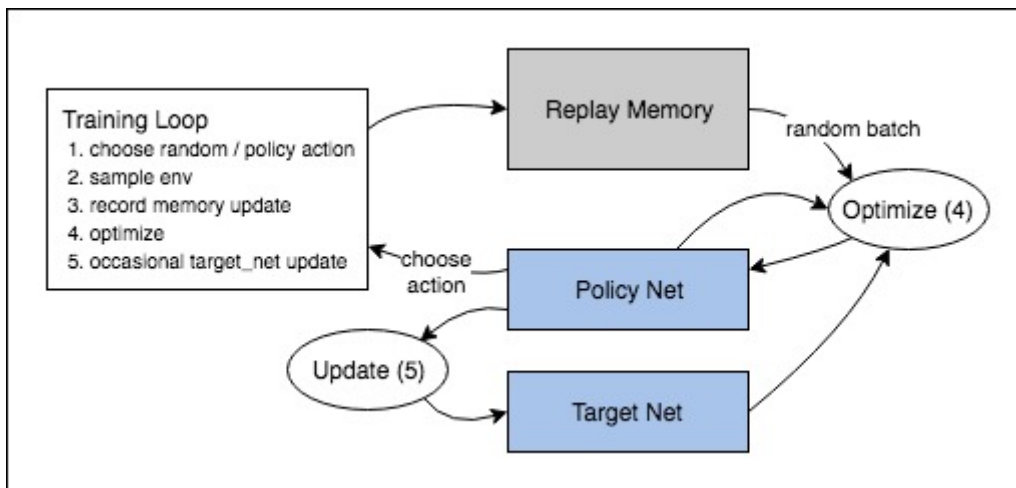
The approximated Q-value function Q^* for a state s a action a is defined as

$$Q^*(s, a) = \left[r + \gamma \max_{a'} Q^*(s', a') \right] | s, a$$

↑ Current state-action pair
 ↑ Reward
 ↑ Discount factor
 ↑ Estimated state-action pair

1. **Current state-action pair** : A state obtained through the environment/simulation.
2. **Reward**: The reward obtained by taking action " a " from state " s ".
3. **Discount factor**: Used to define how much future rewards should account in the calculation of Q-value.
4. **Estimated state-action pair**: From state s we transform to s' by taking action a . Now a' is the most probable action (best) that can be taken from s' .

The overall resulting training process looks as follows:



We need to create two neural networks a policy network ($Q(S, A)$) and a target network ($Q(S', A')$). Using the environment, we collect State (s), Action (a), Reward, next state (s').

```
# first setup simulation enviroment.
env = gym("Breakout-v0")

# get an actio from the policy network using eplsilon greedy algorithm
action = policy_net.get_action_using_epsilon_greedy_method()

# Use generated action to collect reward(R) and next state(S')
next_state, reward, terminal, _ = env.step(action)
```

Now we have collected all the values required to calculate the new Q-value

```
# Use Policy network to get action(a) using state(s)
# Policy Network is the agent we are training. We expect this network to return
us
# q_values that helps us take right action
q_value = policy_net(state)
q_value = q_value[action]

# The Target network is the network we use to estimate the state-action pair.
target_value = target_network(next_state).max()
updated_target_value = reward + (discount_factor * target_value * (1-terminal))
```

The target-value will act as a label to out policy network. We use the huber-loss to calculate the difference and run the optimization step:

```
loss = huber_loss(q_value, updated_target_value)
```

We need two networks the policy and target network to make sure that the training is stable across episodes. Every few episodes the target network is updated with weights from the policy network.

The DQN reinforcement learning algorithm shows that maximizing reward is enough to implicitly catalyze intelligence in an environment a hypothesis postulated by the paper Reward is Enough by D Silver et el.

Reference text and images taken from:

- <https://towardsdatascience.com/reinforcement-learning-made-simple-part-1-intro-to-basic-concepts-and-terminology-1d2a87aa060>
- <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/>
- <https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56>
- Reinforcement Learning 2nd Edition by Richard Sutton and Andrew Barto
- Playing Atari with Deep Reinforcement Learning by Minh et al.
- Reward is enough D. Silver et al.