Reinforcement Learning

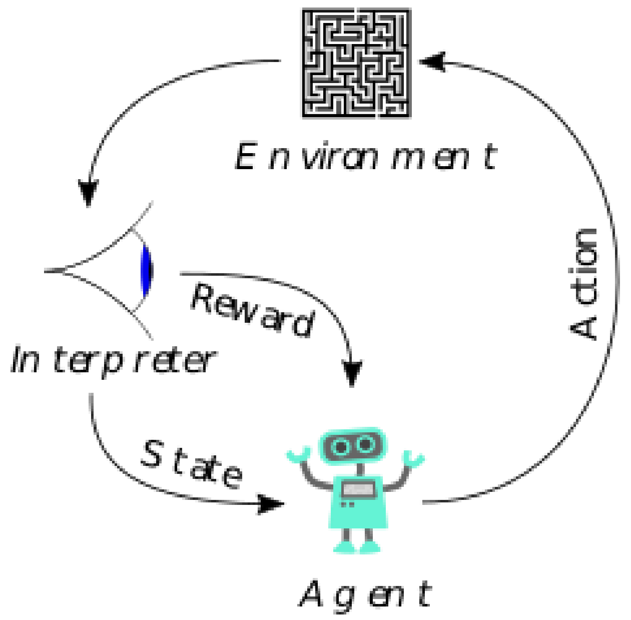
Christian Camilo Urcuqui López, MSc

quantil

matemáticas aplicadas

Introduction

The idea behind this type of learning is how to map situations to actions - so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must **discover** which actions yield the most reward by trying them



Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Introduction

We can mention two characteristics:

- trial-and-error
- search delayed reward

Reinforcement Learning (RL) is defined by characterizing a learning problem through the **interaction with environments**. The objective of this approach is to reach a goal which is reaching by rewards obtained from the environment.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

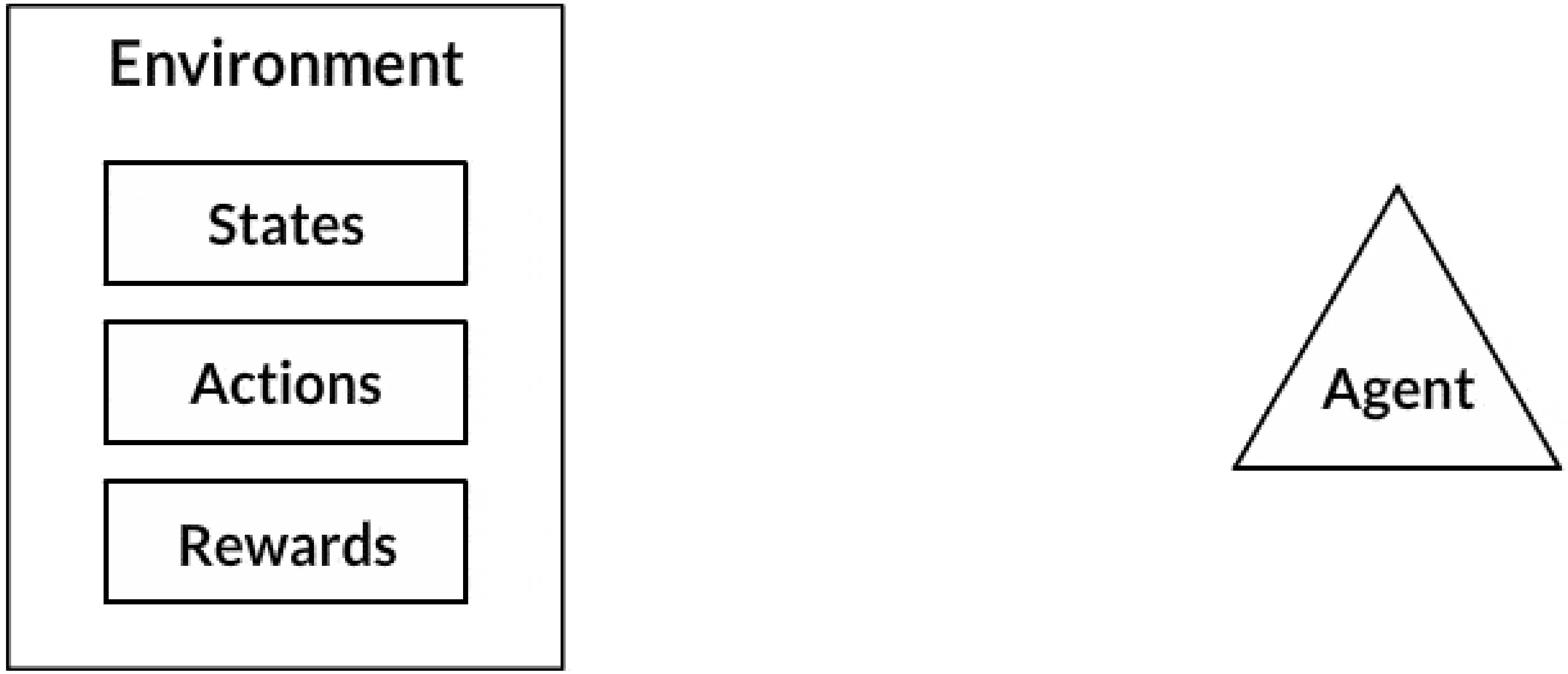
Introduction

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

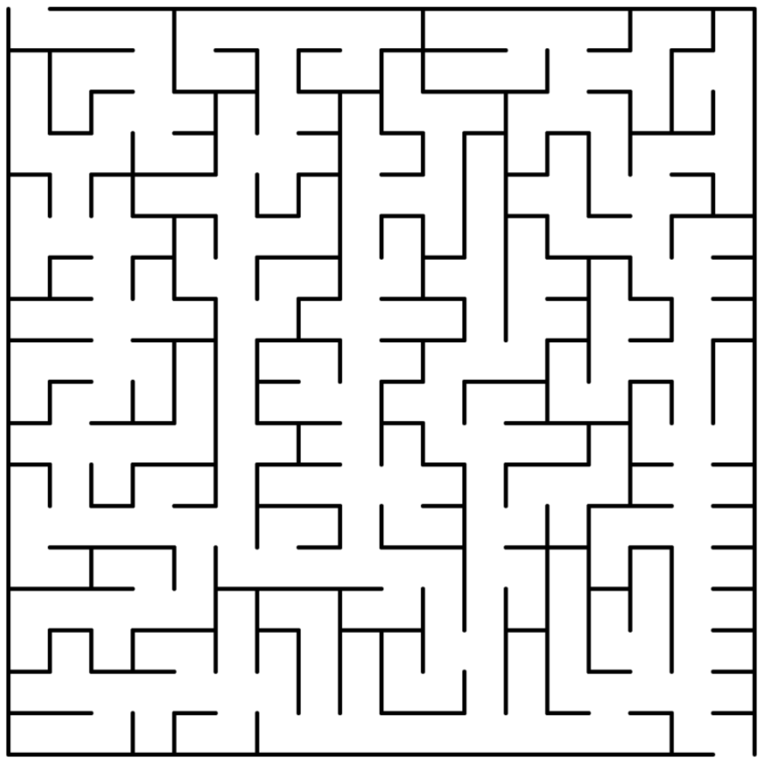
Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in the field of machine learning. Supervised learning is learning from a training set of labeled examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from **interaction**.

In the same way, RL is different from unsupervised learning, which is typically about finding structure hidden in collections of unlabeled data. Although one might be tempted to think of reinforcement learning as a kind of unsupervised learning because it does not rely on examples of correct behavior, reinforcement learning is **trying to maximize a reward signal instead of trying to find hidden structure.**

One of the challenges that arise in RL, and not in other kinds of learning, is the trade-of between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before


Environment
States
Actions
Rewards
Agent

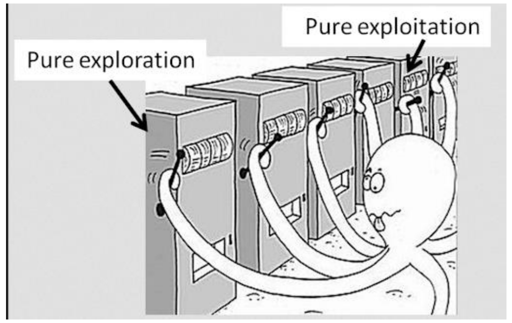
- The agent is an intelligent program
- The environment is the maze
- The state is the place in the maze where the agent is
- The action is the move we take to move to the next state
- The reward is the points associated with reaching a particular state. It can be positive, negative, or zero



Nandy, A., & Biswas, M. (2017). *Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python*. Apress.

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

K-Armed Bandit



The action is to choose and pull one of the levers, and we win a certain amount of money that is the reward associated with the lever (action). The task is to decide which level to pull to maximize the reward. *This is a simplified reinforcement learning problem because there is only one state, or one slot machine, and we need only decide on the action.*

Let us say *Q(a)* is the value of action *a*. Initially, *Q(a) = 0* for all *a*. When we try action *a*, we get reward $r_a \geq 0$. If rewards are deterministic, we always get the same r_a for any pull of *a, we can just set Q(a)=* r_a*, so if we want to explore a higher reward we can choose different actions and store Q(a) for all a, and choose the action with the maximum value*

$$a^* \text{ if } Q(a^*) = \max_a Q(a)$$

If the rewards are not deterministic but stochastic. The amount of of rewards is defined by the probability distribution *p(r | a)*, we define as the estimate value of action *a* at time $Q_t(a)$s an average of all rewards received when action *a* was chosen before time t. An online update can be defined as

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta[r_{t+1}(a) - Q_t(a)],$$

$r_{t+1}(a)$ is the *(t+1)*

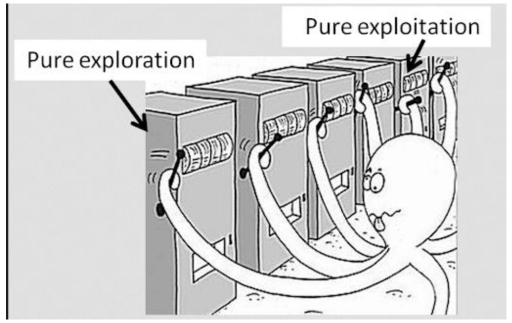
η learning factor (gradually decreased)

r_{t+1} is the desired output

$Q_t(a)$ current prediction

$Q_{t+1}(a)$ is the expected value of action *a* at time t+1 and converges to the mean of p(r | a)

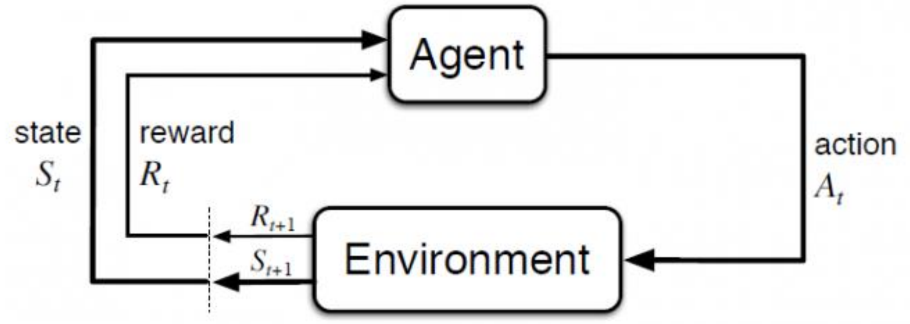
Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

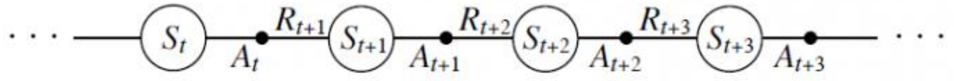


K-Armed Bandit

The full reinforcement learning problem generalizes this simple case in a number of ways. First, we have several states. This corresponds to having several slot machines with different reward pro $p(r|s_i,a_j)$, and we need to lear $Q(s_i,a_j)$ learn which is the value of a_j king action , s_i hen in state . Second, the actions affect not only the reward but also the next state, and we move from one state to another. Third, the rewards are delayed and we need to be able to estimate immediate values from delayed rewards.

"We define a reinforcement learning method as any effective way of solving reinforcement learning problems, and it now clear that these problems are closely related to optimal control problems, particularly stochastic optimal control problems such as those formulated as Markov decision processes (MDP)."




$\cdots$ S_t A_t R_{t+1} S_{t+1} A_{t+1} R_{t+2} S_{t+2} A_{t+2} R_{t+3} S_{t+3} A_{t+3} $\cdots$

- St: State of the agent at time *t*
- At: Action taken by agent at time *t*
- Rt: Reward obtained at time *t*

The overall goal for the agent is to maximize the cumulative reward it in the long run (*finite-horizon*). Total reward at any time instant *t* is given by:

(1) $$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

But, certain tasks are continuing (*infinite-horizon*), in (1) all the rewards have equal weight, the idea is to include a discounting factor γ, 0<γ<1.

(2) $$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

The rewards further in the future are getting diminished. The discount factor γ can be understood as a tuning parameter which can be changed based on how much one wants to consider the long term (γ close to 1) or short term (γ close to 1)

We are going to have two constants, these are gamma (γ) and lambda (λ).

- *Gamma* is used in each state transition and is a constant value at each state change. Gamma allows you to give information about the type of reward you will be getting in every state.
- *Lambda* is generally used when we are dealing with temporal difference problems. It is more involved with predictions in successive states. When lambda increases we can infer that the algorithm is learning fast.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Elements

RL systems have the next subelements: a policy, a reward signal function, a value function, and, optionally, a model of the environment

- **A model** of the environment, it is used for planning, where the model can find information about the resultant next state and the next reward for a current state and action.We can see the problem modeled using a Markov decision process. The reward and next state are sampled from their respective probability distributions $p(r_{t+1}|s_t, a_t)$ and $P(s_{t+1}|s_t, a_t)$.
  - Methods for solving RL problems that use models planning are called model-based methods
  - Methods who don’t use models are known as model-free methods that are trial-and-error learners - viewed as almost the opposite of planning.

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

Elements

RL systems have the next subelements: a policy, a reward signal function, a value function, and, optionally, a model of the environment.

- The sequence of actions from the start to the terminal state is an ***episode,*** *or* ***trial***
- **The policy, π,** defines the agent's behavior and is mapping from the states of the environment to actions: **π : S $\rightarrow$ A.** The policy defines the action to be taken in any state st: at = **π(st).** The value of a policy **π** $V^{\pi}(s_t)$, is the expected cumulative reward that will be received while the agent follows the policy, starting fro

$$V^{\pi}(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process.

Optimal policy $V^*(s_t) = \max_{\pi} V^{\pi}(s_t), \forall s_t$

Elements

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

In some applications, for example, in control, instead of working with the values of states, $V(s_t)$, we prefer to work with the values of state-action pairs, $Q(s_t,a_t)$.

- $V(s_t)$, it denotes how good it is the agent to be in state s_t
- $Q(s_t,a_t)$, it denotes how good is to perform an action a_t when in state s_t

So, we define the $Q^*(s_t,a_t)$ as the value, that is, the expected cumulative reward. The value of a state is equal to the value of the best possible action:

$$\begin{aligned} V^*(s_t) &= \max_{a_t} Q^*(s_t,a_t) \\ &= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right] \\ &= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1}\right] \\ &= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \\ V^*(s_t) &= \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t,a_t) V^*(s_{t+1}) \right) \end{aligned}$$

Bellman's equation

To each possible next state s_{t+1}, we move with probability $P(s_{t+1}|s_t,a_t)$ and continuing from there using the optimal policy, the expected cumulative reward is $V^*(s_{t+1})$ .We sum over all such possible next states, and discount it because it is one time step later. Adding our immediate expected reward, we get the total expected cumulative reward for action ***a_t***

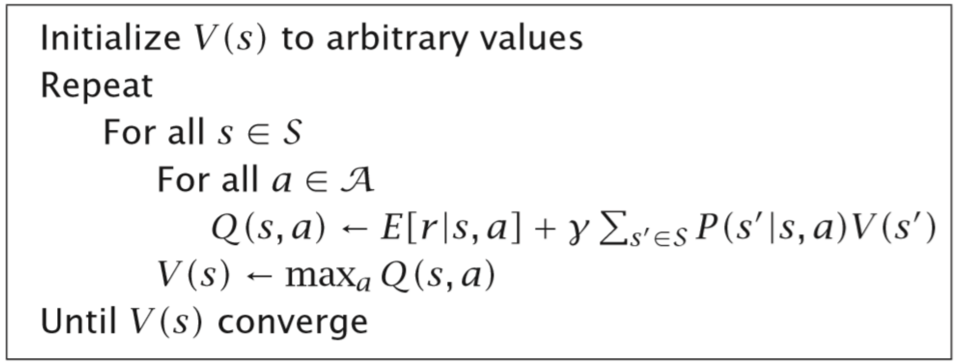
Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

Elements

- **A reward signal function defines the goal of a RL problem.** On each time step, the environment sends to the agent a reward, the agent's sole objective is to maximize the total reward it gets over the long run. It indicates what is good in an immediate sense.
- **A value function specifies what is good in the long run**. Roughly speaking, it is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Elements

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.



```
Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge
```

Figure 18.2 Value iteration algorithm for model-based learning.

Once we have $Q^*(s_t, a_t)$ values, we can then define our policy **π** as taking the action a_t^* which has the highest value among all $Q^*(s_t, a_t)$

$$\pi^*(s_t) : \text{Choose } a_t^* \text{ where } Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t)$$

We can use a greedy search at each local step we get the optimal sequence of steps that maximizes the cumulative reward

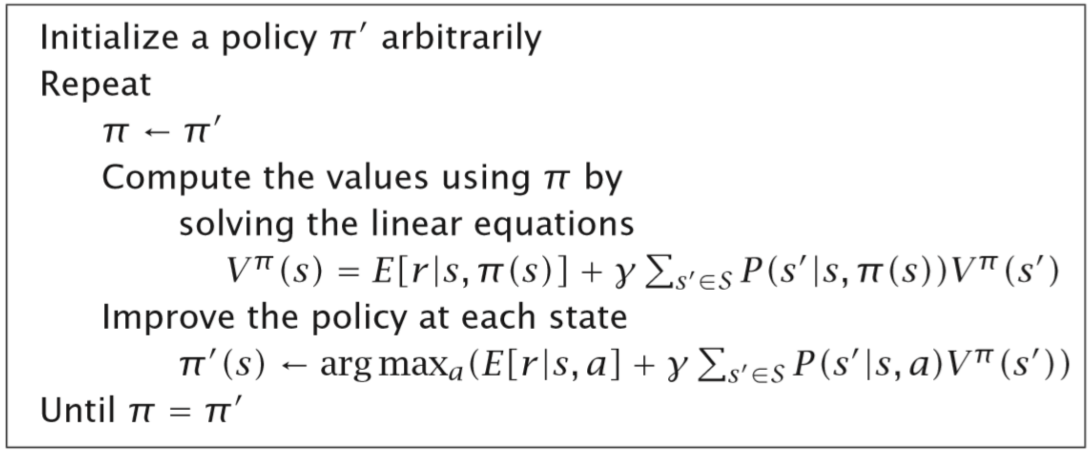
https://en.wikipedia.org/wiki/Greedy_algorithm

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

Model-based learning

We completely know the environment model parameters, $P(s_{t+1}|s_t, a_t)$ and $p(r_{t+1}|s_t, a_t)$. Once we have the optimal value function, the optimal policy is to choose the action that maximizes the value in next state:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1}|s_t, a_t] + \gamma \sum_{s_{t+1}\in S} P(s_{t+1}|s_t, a_t) V^*(s_t + 1) \right)$$



```
Initialize a policy $\pi'$ arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using $\pi$ by
        solving the linear equations
            $V^\pi(s) = E[r|s, \pi(s)] + \gamma \sum_{s'\in S} P(s'|s, \pi(s)) V^\pi(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s'\in S} P(s'|s, a) V^\pi(s'))$
Until $\pi = \pi'$
```

Figure 18.3 Policy iteration algorithm for model-based learning.

Model-based and model-free RL

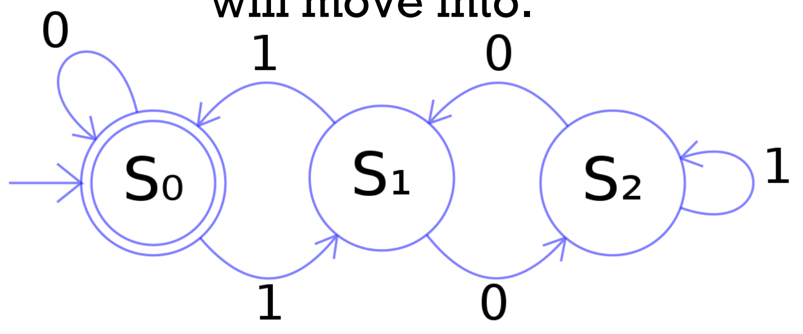
- **Model-based RL algorithms** (namely value and policy iterations) work with the help of a transition table. **A transition table can be thought of as a life hack book which has all the knowledge the agent needs to be successful in the world it exists in**. Naturally, writing such a book is very tedious and impossible in most cases which is why model dependent learning algorithms have little practical use.
- **Model-free**: **The agent doesn't bother with the MDP model and instead attempts to develop a control function that looks at the state and decides the best action to take.** In that case, the parameters to be learned are the ones that define the control function

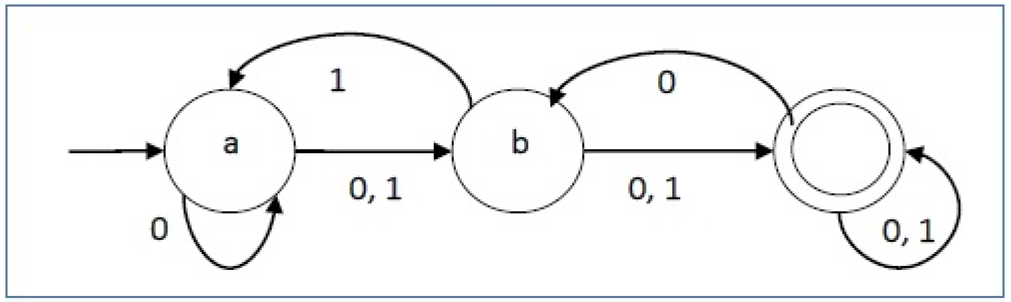
Elements

An **agent** is a software program that make intelligent decisions. Agents should be able to perceive what is happening in the environment.

RL environments

- **Deterministic**, we know the outcome based on the current state
  - ***Deterministic Finite Automata (DFA)***, a finite number of steps and one action for them
  - ***Nondeterministic Finite Automaton***, we don't know exactly which state a machine will move into.





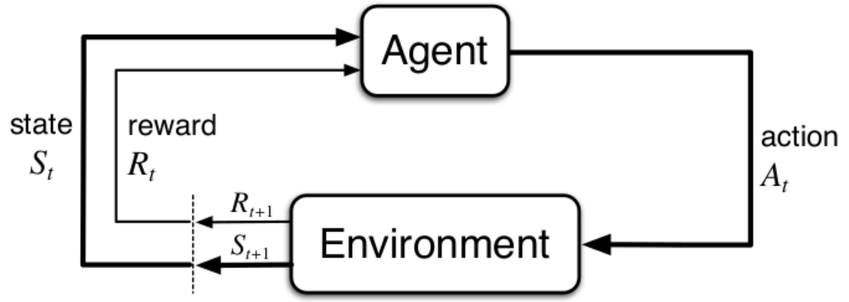
Elements

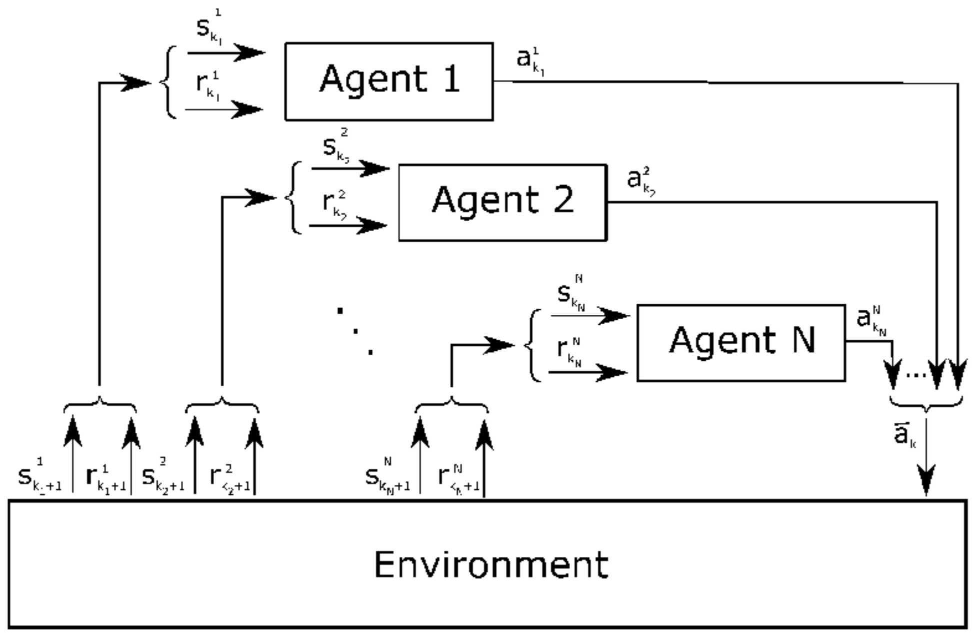
- **Stochastic environment**, we cannot determine the outcome based on the current state. There will be a greater level of uncertainty.
- **Fully observable environment**, when an agent can determine the state of the system at all times. For example, chess.
- **Partially observable environment**, when an agent cannot determine the state of the system at all times. For example, poker game.
- **Discrete environment**, there is only a finite state of actions actions available for moving from one state to another.
- **Continuous environment**, there is an infinite state of actions available for moving from one state to another.
- **Episodic and non-episodic environment**, in an episodic, an agent’s current action will no affect a future action, whereas in the non-episodic environment all agents actions are related and is also called **sequential environment**

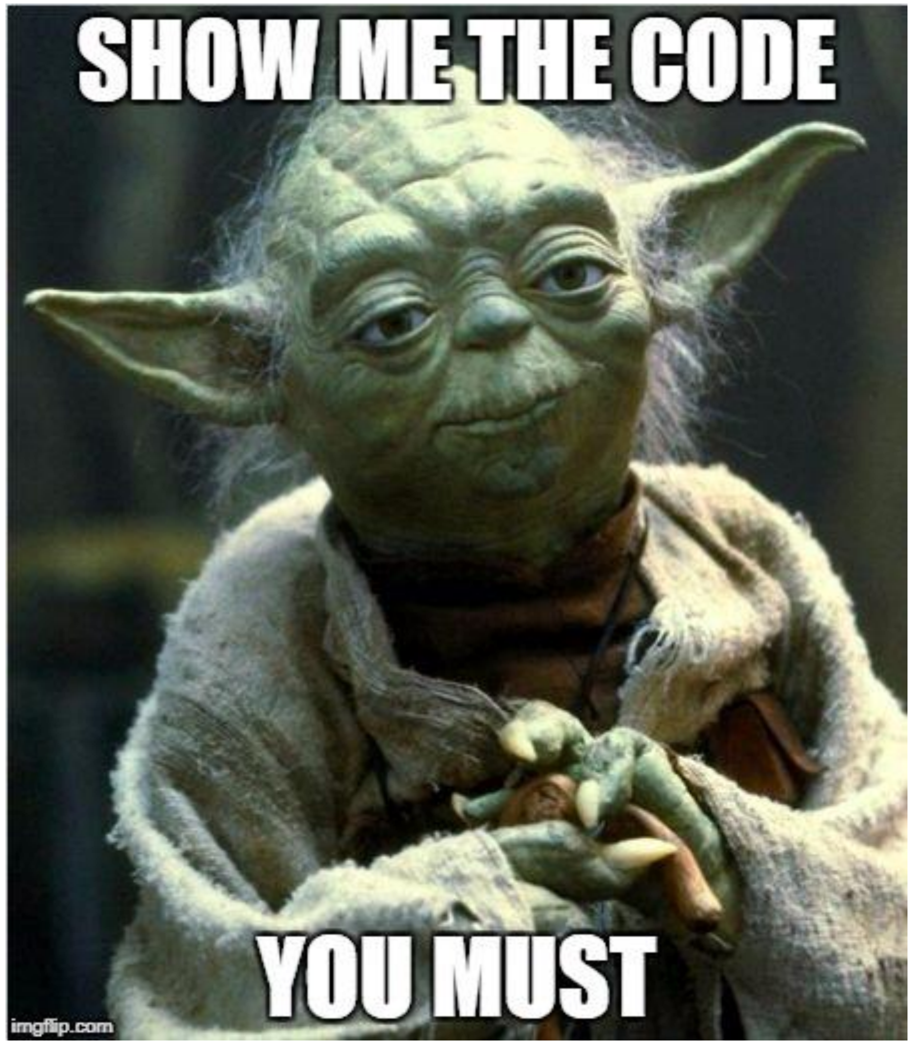
Single and multi-agent environment

When we are dealing with complex problems, we use multi-agent reinforcement learning. Complex problems might have different environments where the agent is doing different jobs to get involved. There will be different agents acting in completely different environments and they can communicate to each other.

A multi-agent environment will be mostly stochastic it has a greater level of uncertainty.







https://github.com/urcuqui/Data-Science/blob/master/Machine_learning/Reinforcement%20Learning/Reinforcement%20Learning.ipynb

SARSA

SARSA stands for State Action Reward next State and next Action. It is another RL approach where is derived from temporal difference learning

Temporal Difference Learning

It is a model-free RL algorithm. This means that the agent learns through actual experience rather than through a readily available all-knowing-hack-book (transition table). The idea is to predict the best path over a period of time. The agent has no idea about the reward and transition systems.

Let's suppose that we stay at S0 and we want to go to SF, we are going to get rewards in each state and the TD updates the knowledge of the agent rather than on every episode.

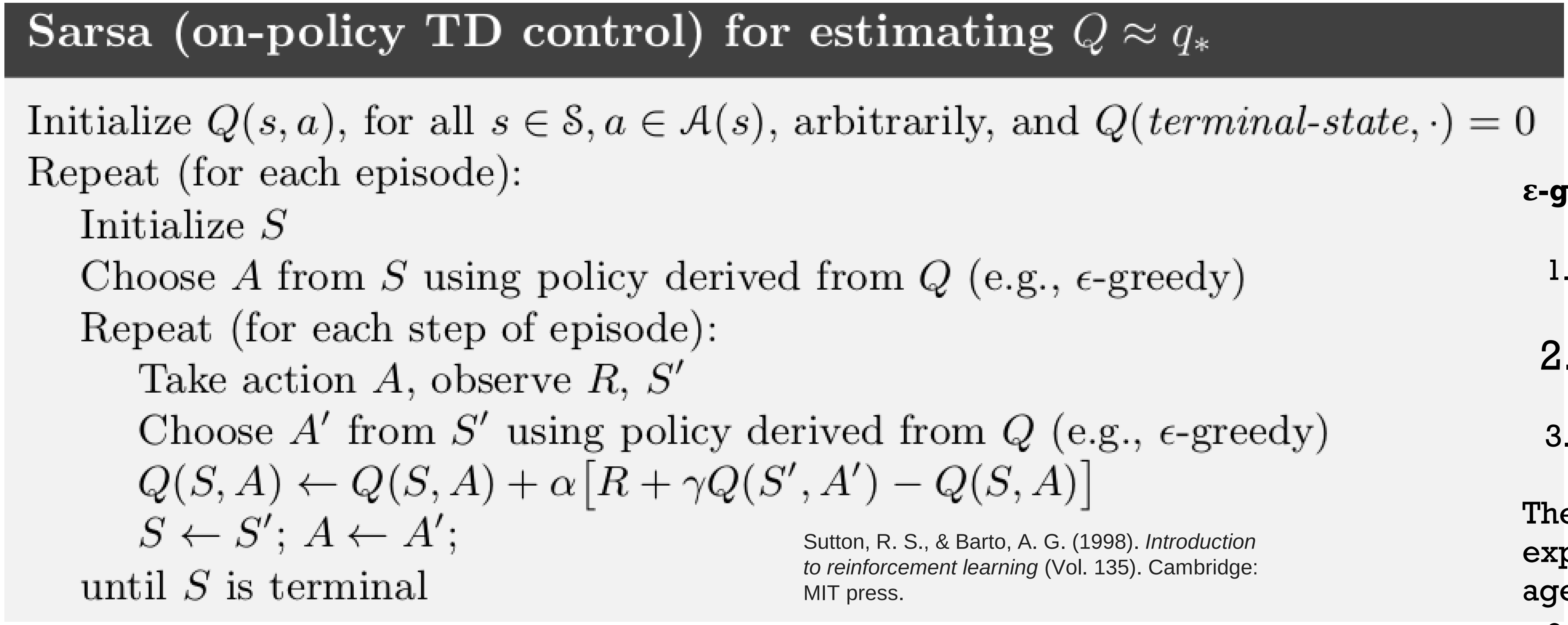
$$\begin{tikzcd} S_0 \arrow[r, "r_0"] & S_1 \arrow[r, "r_1"] & S_2 \arrow[r, "r_2"] & S_F \end{tikzcd}$$

Figure 2-49. State transition

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize}\Big[\text{Target} - \text{OldEstimate}\Big]$$

- Targed- oldEstimate is called the *target error*
- StepSize is usually denoted as α and called the *learning rate* (0-1)



Sarsa (on-policy TD control) for estimating $Q \approx q_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (Vol. 135). Cambridge: MIT press.

ε-greedy policy

1. Generate a random number r ∈ [0,1]
2. if r<ε choose an action derived from the Q values
3. Else choose a random action

The value of ε determines the exploration-exploitation of the agent.

- if ε is large, less exploration, more exploitation
- if ε is small, the agent will explore the environment even more

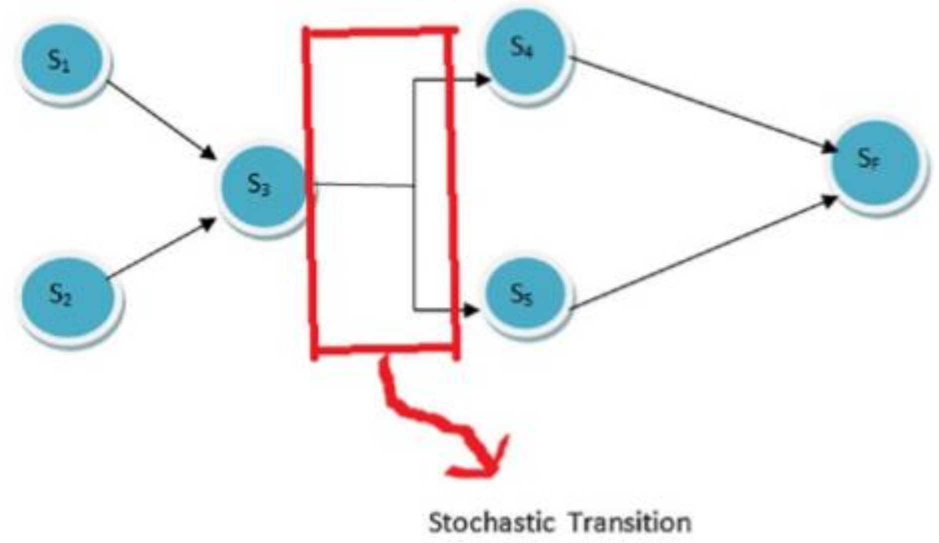
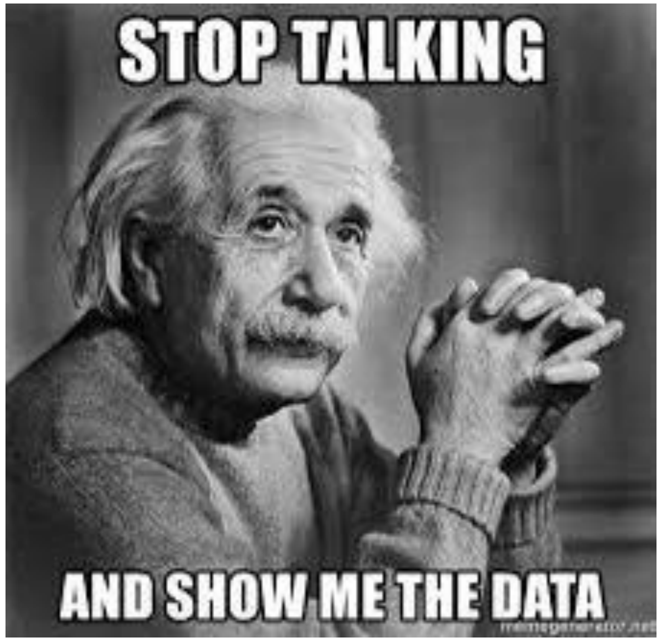


Figure 2-50. The Markov Chain

SARSA is known as an own policy RL. An own policy means that we can see only our own experiences.

Ivan Pavlov - Pavlovian or Classical Conditioning

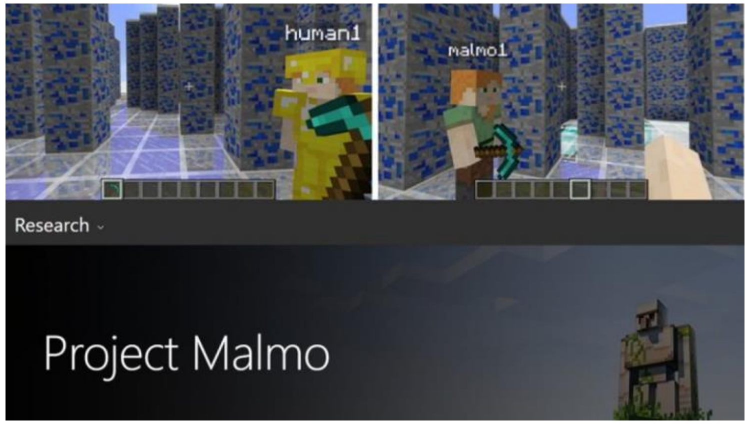


https://github.com/urcuqui/Data-Science/blob/master/Machine_learning/Reinforcement%20Learning/Reinforcement%20Learning.ipynb

OpenAI

Google DeepMind

I Use RL Glue

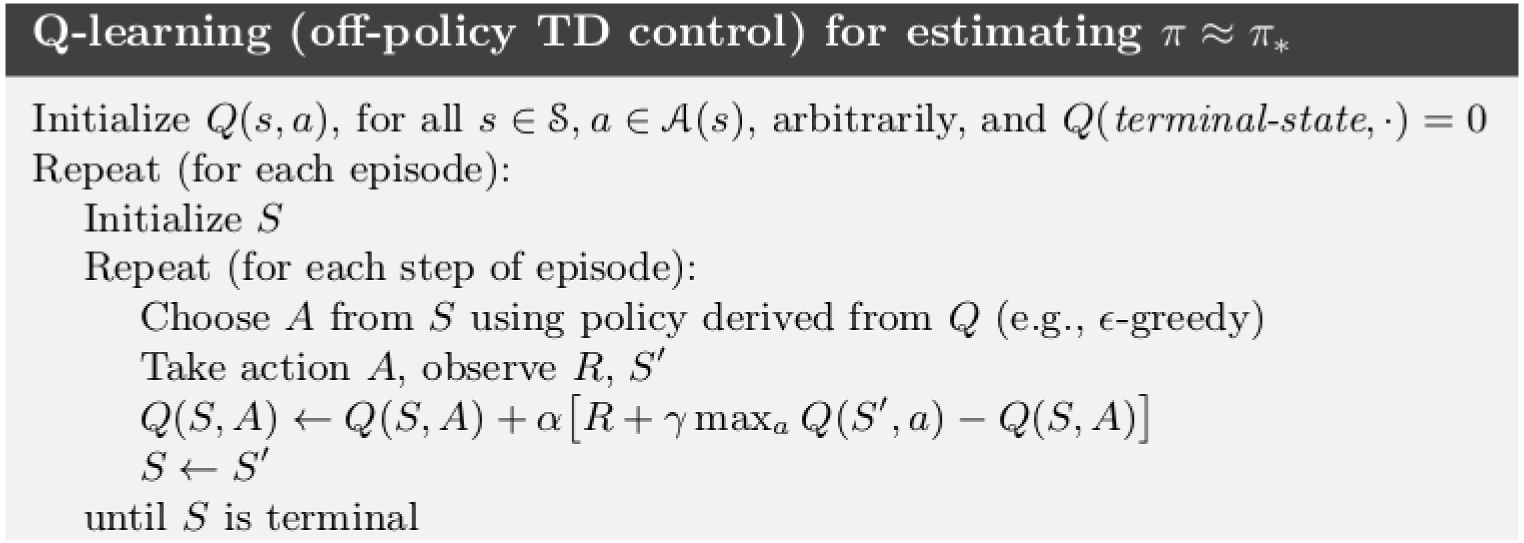


Microsoft

aws DEEPRACER LEAGUE

Q-Learning

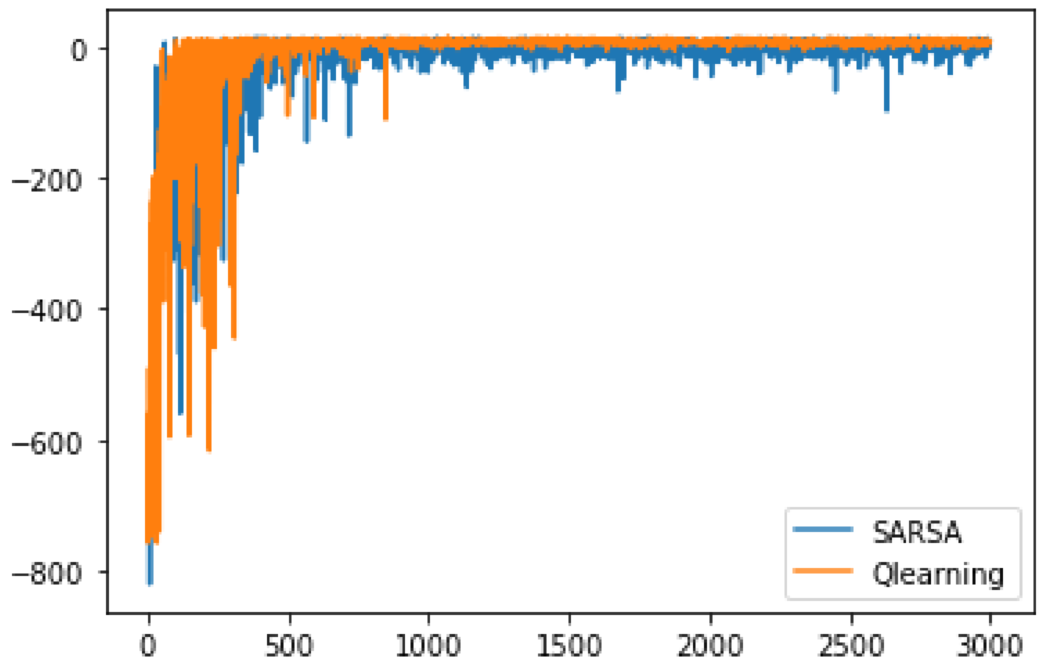
It is an off-policy TD control policy. The difference between this and SARSA is that it doesn't follow a policy to find the next action A' but rather chooses the action in a greedy fashion



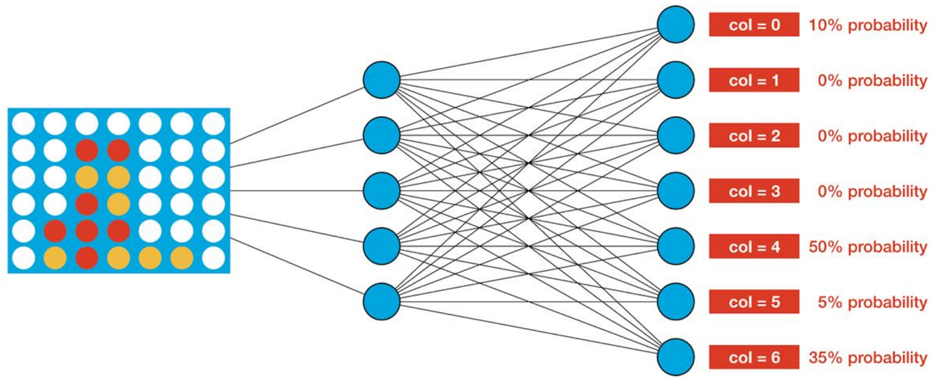
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

https://github.com/urcuqui/Data-Science/blob/master/Machine_learning/Reinforcement%20Learning/Reinforcement%20Learning.ipynb


0
−200
−400
−600
−800
0
500
1000
1500
2000
2500
3000
SARSA
Qlearning

Deep Reinforcement Learning



https://www.kaggle.com/alexisbcook/deep-reinforcement-learning