Synthetic Intelligence Lab

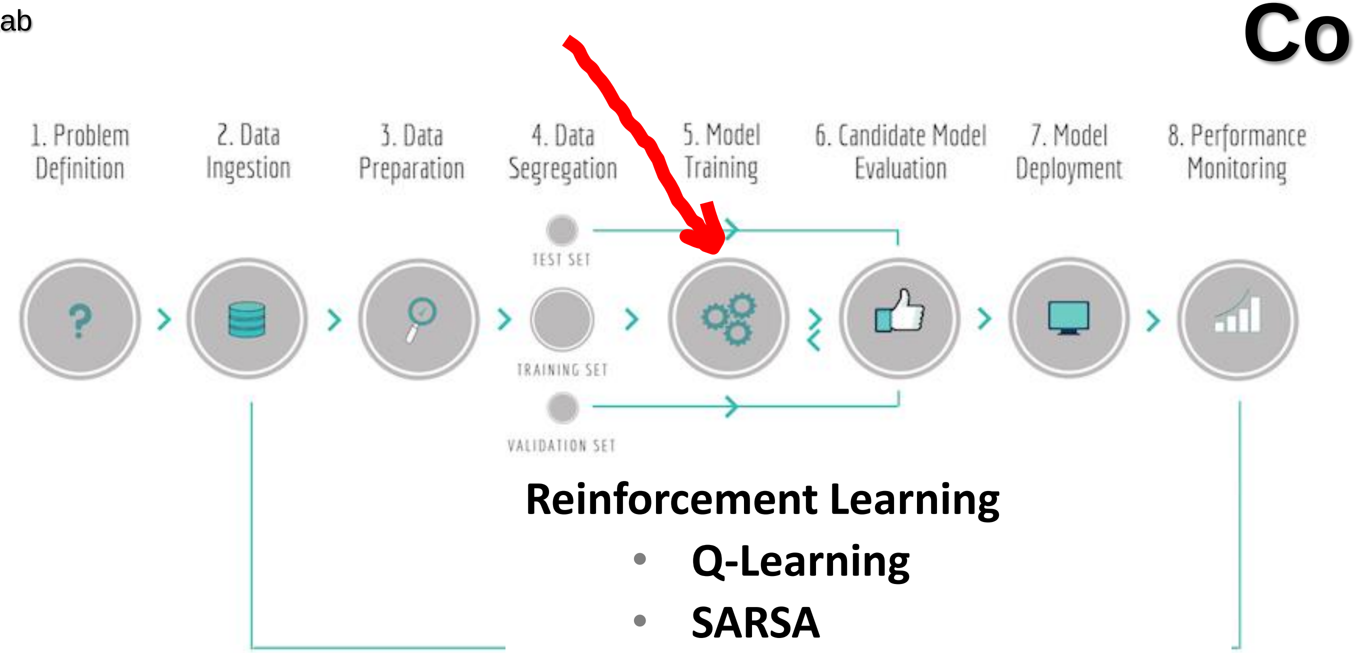
Universidade do Minho
Departamento de
Informática

DADOS e APRENDIZAGEM AUTOMÁTICA

Reinforcement Learning

MESTRADO (integrado) EM ENGENHARIA INFORMÁTICA

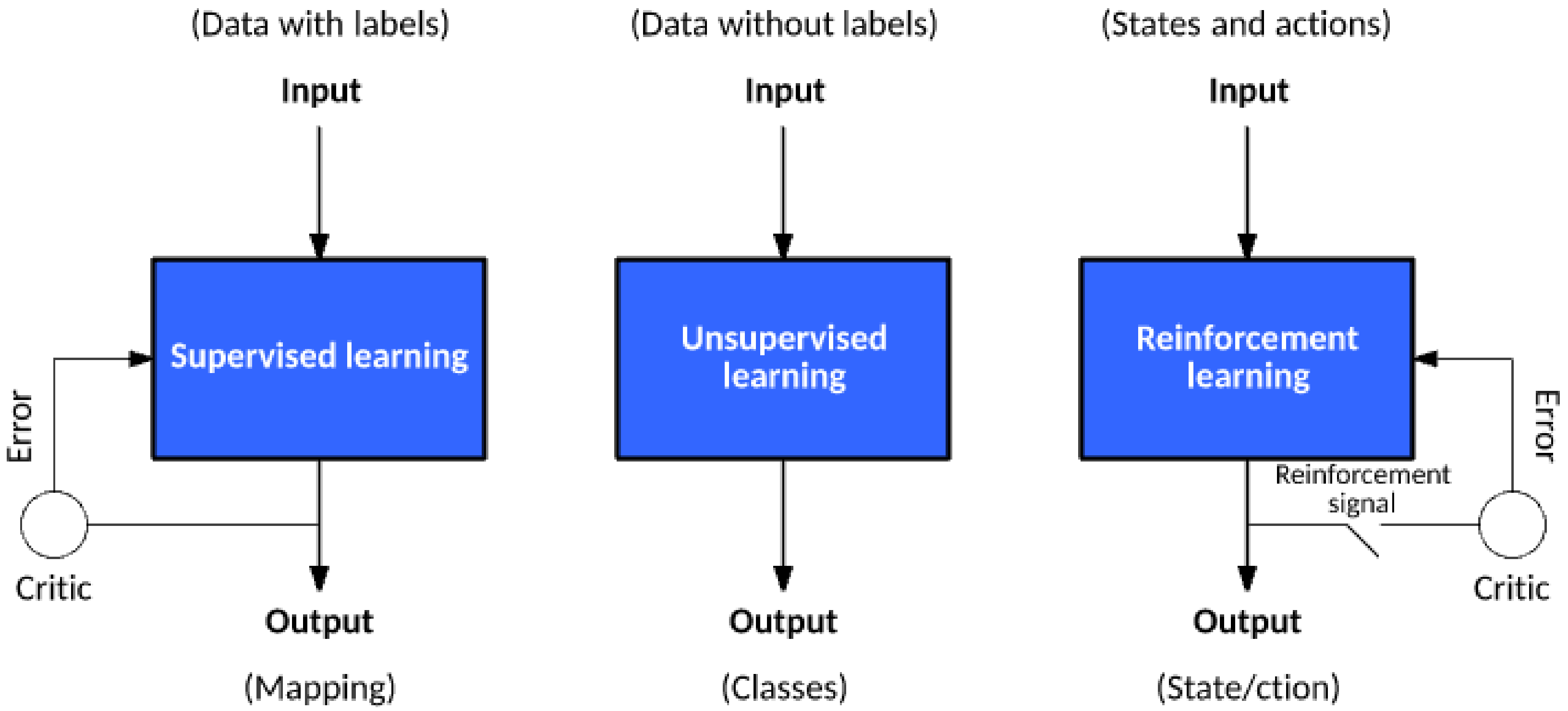
Contents



Reinforcement Learning

- **Q-Learning**
- **SARSA**

Supervised, Unsupervised and Reinforcement Learning



(Data with labels)

Input

Supervised learning

Error

Critic

Output

(Mapping)

(Data without labels)

Input

Unsupervised learning

Output

(Classes)

(States and actions)

Input

Reinforcement learning

Error

Reinforcement signal

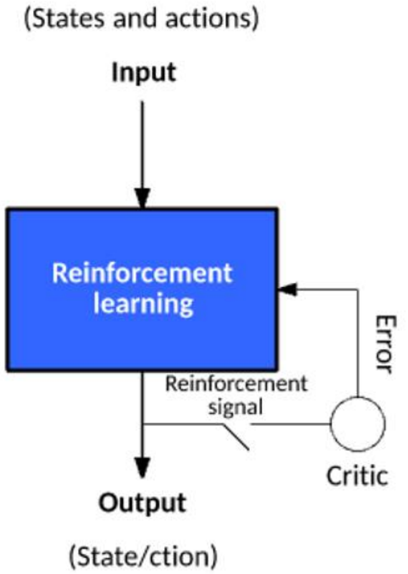
Critic

Output

(State/ction)

Reinforcement Learning

- Reinforcement Learning is learning "what to do" in order to maximise a **numerical** reward value:
  - The learner is not told which actions to perform;
  - The learner must discover which actions guarantee the greatest return by **trying them out**.
- The essential characteristics are:
  - **Trial-and-error**;
  - **Delayed reward**.
- Complementary characteristics are:
  - Time;
  - There is a learner, but no teacher!



(States and actions)

Input

Reinforcement learning

Error

Reinforcement signal

Critic

Output

(State/ction)

- Reinforcement learning **is not defined** by the characterization of learning methods (**we do not program algorithms to learn**);
- Reinforcement learning **is defined** by the characterization of the learning problem (**we program the characteristics of the problem to learn about**);

- Reinforcement Learning **is not looking** at **past experiences** to make decisions (CBR, ANN);
- Reinforcement Learning **is looking** at the **current state** and deciding what to do, predicting the expected future;

Applications

Applications

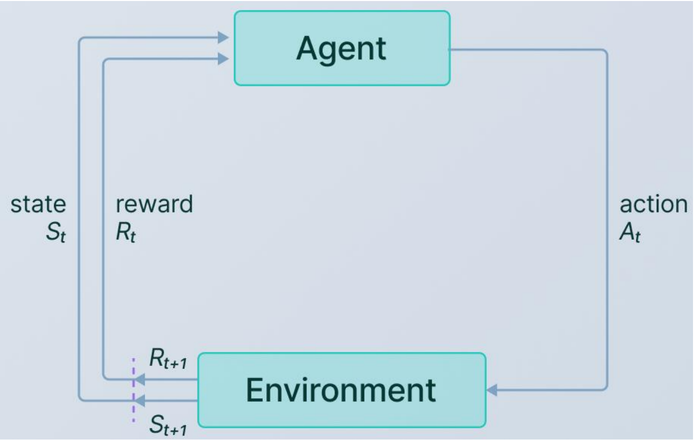
- **Robotics** for industrial automation.
- Business **strategy planning**
- Machine learning and data processing
- It helps you to create training systems that provide custom instruction and materials according to the requirement of students.
- Aircraft control and **robot motion** control
- Autonomous cars

When not to use RL?

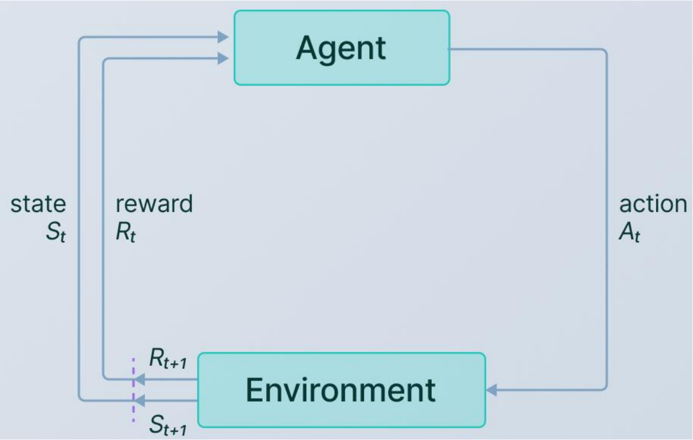
- When you **have enough data** to solve the problem with a supervised learning method
- You need to remember that Reinforcement Learning is **computing-heavy** and **time-consuming**. in particular when the **action space is large**.

Terms

- **Agent (A)**: An assumed entity which performs actions in an environment to gain some reward.
- **Environment (e)**: A scenario that an agent has to face.
- **Reward (R)**: An immediate return given to an agent when he or she performs specific action or task.
- **State (S)**: State refers to the current situation returned by the environment.
- **Policy (π)**: Strategy to be applied by the agent to decide the next action based on the current state.
- **Value (V)**: Expected long-term return with discount, as compared to the short-term reward.
- **Value Function**: Specifies the value of a state that is the total amount of reward. It corresponds to an agent which should be expected beginning from that state.
- **Model of the environment**: Mimics the behavior of the environment. It helps you to make inferences to be made and also determine how the environment will behave.
- **Model based methods**: (e.g. Q-Learning) It is a method for solving reinforcement learning problems which use model-based methods.
- **Q value or action value (Q)**: Q value is quite similar to value. The only difference between the two is that it takes an additional parameter as a current action.



Procedure



- Procedure:
  - Agent performs action on the environment;
  - Environment assigns reward/penalty;
  - Agent calculates the convenience of the action;
  - Agent performs new action on the new environment.

- The life cycle produces a sequence of states S_i, actions A_i and immediate rewards R_i:

$$\begin{tikzcd} S_0 \arrow[r, "A_0", "R_0"'] & S_1 \arrow[r, "A_1", "R_1"'] & S_2 \arrow[r, "A_2", "R_2"'] & \ldots \end{tikzcd}$$

- Each time the agent performs an action on the environment, it is assigned a reward or a penalty demonstrating the desirability of his action.

- The agent's task is to **learn a control policy**, $\pi: S \to A$, that **maximises the (expected) sum of rewards**, with future rewards discounted exponentially by their delay:

$$R_0 + \gamma^1 R_1 + \gamma^2 R_2 + \ldots \text{, with } 0 \leq \gamma < 1 \qquad \sum_{i=0}^{n} \gamma^i R_i$$

Exploitation/Exploration trade-off

One of the challenges that arise in *reinforcement learning* is the **trade-off between exploration and exploitation**.

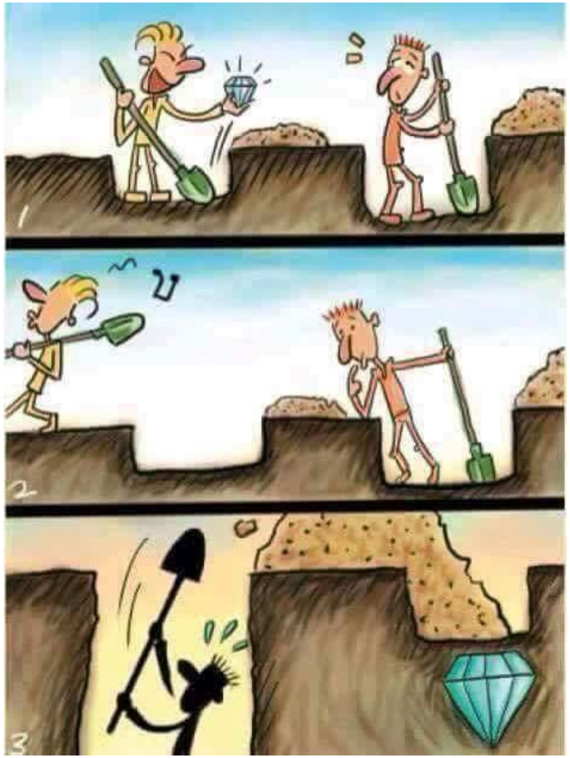
Exploration : is about finding more information about the environment. (in other words) exploring a lot of states and actions in the environment.

Exploitation : is about exploiting the known information to maximize the reward.

If you select a **greedy action**, you are ***exploiting*** **your current knowledge** of the values of the actions. If instead you select one of the **non-greedy actions**, then you are ***exploring*** because this enables you to improve your estimate of the non-greedy action's value.

Exploration/Exploitation

The agent should prefer actions that he knows are effective – exploit;
But to discover such actions, he has to try actions not selected before – explore.



Q-Learning, State-Action-Reward-State-Action (SARSA)

We want to obtain a function Q(S,A) (action-value-function) that predicts the best action A in state S in order to maximize a cumulative reward:

Q-learning (off-policy/greedy learner):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

New estimation

Learning rate

Discount factor (=0.9)

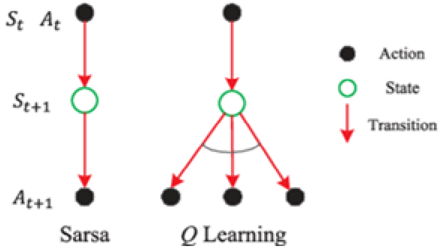
SARSA (on-policy learner):

$$\underset{a_{t+1}}{Q(s_t, a_t)} \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, \underset{s_{t+1}}{a_{t+1})} - Q(s_t, a_t)]$$

Q-Learning - The agent is in state 1, performs action 1 and gets reward 1 (r_{t+1});
It sees the maximum reward in state 2 and updates the value of action 1 performed in state 1.

SARSA - The agent is in state 1, performs action 1 and gets reward 1 (r_{t+1});
In state 2, it performs action 2 and obtains reward 2, and then updates the value of action 1 in state 1;

SARSA very much resembles Q-learning, the key difference is that SARSA learns the Q-value based on the action performed by the current policy instead of the greedy policy



Discount factor - Encourages the agent to prefer immediate rewards to delayed ones;

Q-Learning, SARSA

Q-table - The agent must **base its actions on previously experienced rewards**. It does this by keeping a table with Q-values. It stores each **Q-value for a state-action pair**. **How the agent updates the Q-value is where the difference between SARSA and Q-learning lies**. Depending on the action selection strategy, it can choose the action that results in the highest expected reward.

Action selection - Action selection is **the strategy where the agent bases its selection of actions on**. The most basic strategy is the greedy strategy, which **always goes for the highest reward**. In other words, it always exploits the action with the highest estimated reward. However, chances are that this action selection strategy overlooks possible better actions.

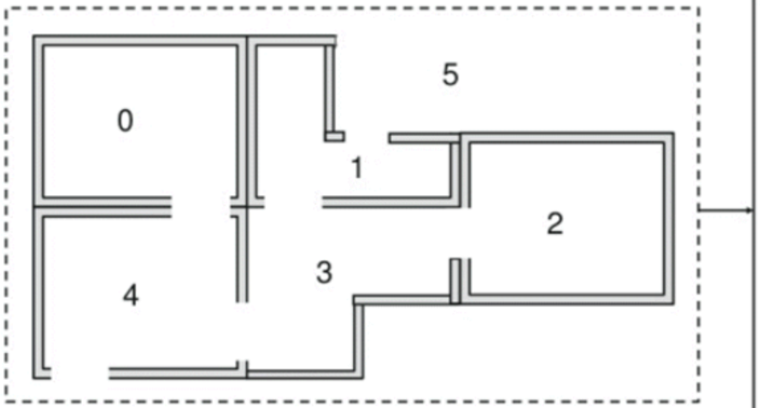
Another strategy is **ε-greedy** (pronounced as epsilon-greedy). ε-greedy is a strategy that balances exploration and exploitation based on the epsilon parameter. Instead of always selecting the action with the best estimated value from the Q-table, it gives a small probability (the epsilon) to select possible actions uniformly and randomly. This makes sure the agent doesn't overlook actions that potentially result in higher rewards.

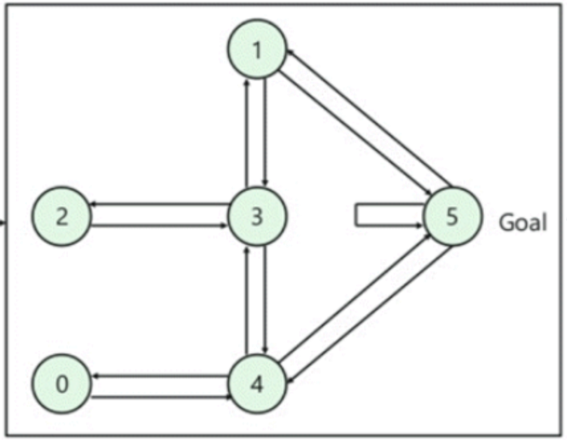
Parameters - Both algorithms need the following parameters:

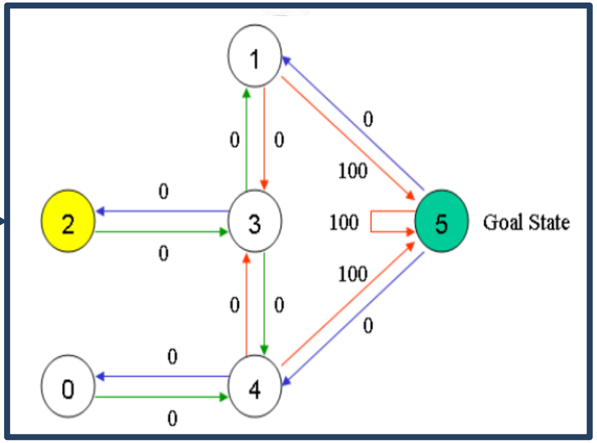
epsilon (ε): the chance that an agent chooses an exploration step
learning rate (α): the rate at which the agent learns from new observations
discount value (γ): the amount future rewards get discounted because direct rewards are more important than future rewards

Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$







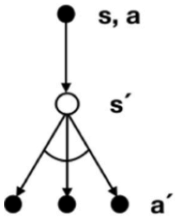
- Rooms numerated from 0 to 4
- Outside area numerated 5
- Doors lead to certain spaces

- Doors conected both ways
- The goal is the outside area

- Doors that connect directly the outside - reward: 100 points
- The other doors - reward: 0 points

For example, an agent traverse from room number 2 to 5:

- Initial state = state 2
- State 2 -> state 3
- State 3 -> state (2, 1, 4)
- State 4 -> state (0, 5, 3)
- State 1 -> state (5, 3)
- State 0 -> state 4
- State 4 -> state (0, 3, 5)

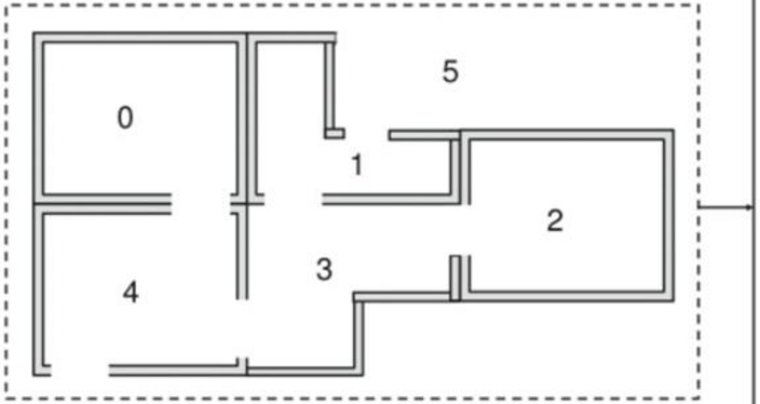


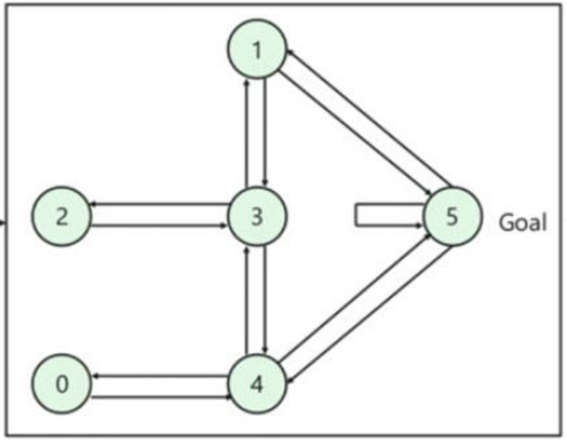
ISLab
Synthetic Intelligence Lab

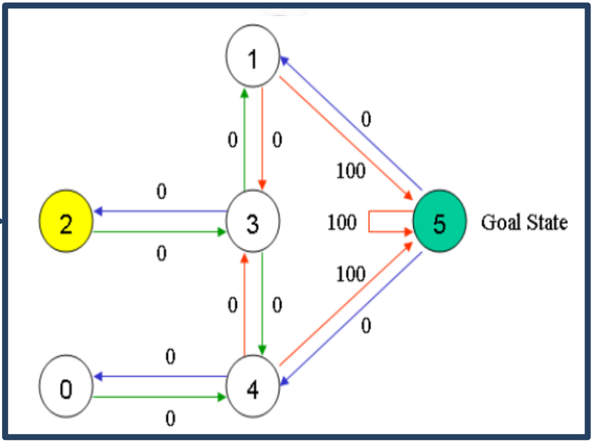
Universidade do Minho
Departamento de
Informática

Q-Learning

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \boxed{\max_a} Q(s_{t+1}, a) - Q(s_t, a_t)]$$







- Rooms numerated from 0 to 4
- Outside area numerated 5
- Doors lead to certain spaces

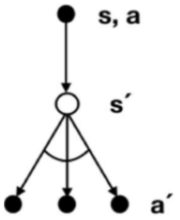
- Doors conected both ways
- The goal is the outside area

- Doors that connect directly the outside - reward: 100 points
- The other doors - reward: 0 points

For example, an agent traverse from room number 2 to 5:

- Initial state = state 2
- State 2 -> state 3
- State 3 -> state (2, 1, 4)
- State 4 -> state (0, 5, 3)
- State 1 -> state (5, 3)
- State 0 -> state 4
- State 4 -> state (0, 3, 5)

Path decided on the maximum Q-value



Q-Learning

Q-Learning algorithm:

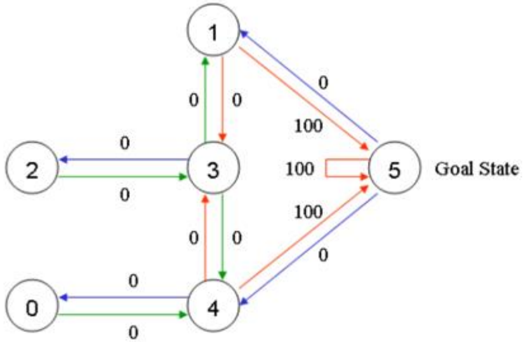
1. **Set** the gamma parameter and environment rewards in the Reward Table.
2. **Initialize** Q-table to zero.
3. **For** each episode:

    Select a random initial state.

    Do While the goal state hasn't been reached.

    - Select one among all possible actions for the current state.
    - Using this possible action, consider going to the next state.
    - Get maximum Q value for this next state based on all possible actions.
    - Compute: *Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*
    - Update Q-table
    - Set the next state as the current state.

We'll start by setting the value of the learning parameter Gamma = 0.8



Action

$$R= \begin{array}{c} \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

In the Reward table, the -1's represent null values (i.e.; where there isn't a link between nodes).

Q-Learning

Q-Learning algorithm:

1. **Set** the gamma parameter and environment rewards in the Reward Table.
2. **Initialize** Q-table to zero.
3. **For** each episode:
    Select a random initial state.
    Do While the goal state hasn't been reached.
    - Select one among all possible actions for the current state.
    - Using this possible action, consider going to the next state.
    - Get maximum Q value for this next state based on all possible actions.
    - Compute: *Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*
    - Update Q-table
    - Set the next state as the current state.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

We'll start by setting the initial state as Room 1.

$$\begin{array}{ccc} & & \text{Action} \\ & \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ R= & \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

From the second row (state 1) of matrix R. There are two possible actions for the current state 1: go to state 3, or go to state 5. By random selection (all Q=0), we select to go to 5 as our action.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cancel{100} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow 100 \\ \\ \\ \\ \\ \end{array}$$

The next state, 5, now becomes the current state. Because 5 is the goal state, we've finished one episode. Our agent's *brain* now contains an updated matrix Q

Q-Learning

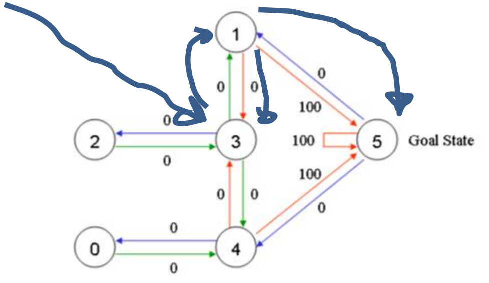
```
Q-Learning algorithm:

1. Set the gamma parameter and environment rewards in the Reward Table.
2. Initialize Q-table to zero.
3. For each episode:
      Select a random initial state.
      Do While the goal state hasn't been reached.
         • Select one among all possible actions for the current state.
         • Using this possible action, consider going to the next state.
         • Get maximum Q value for this next state based on all possible actions.
         • Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
         • Update Q-table
         • Set the next state as the current state.
```

$$Q=\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & \cancel{100} \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

← 100

For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state.

$$R=\begin{array}{c|cccccc} \text{State} \backslash \text{Action} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & -1 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$



Look at the fourth row of matrix R; it has 3 possible actions: go to states 1, 2, or 4. By random selection, we select to go to state 1 as our action.

Now we imagine that we are in state 1. Look at the second row of reward matrix R (i.e. state 1). It has 2 possible actions: go to state 3 or state 5. Then, we compute the Q value:

*Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]*

*Q(3, 1) = R(3, 1) + 0.8 * Max[Q(1, 3), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80*

We use the updated matrix Q from the last episode. Q(1, 3) = 0 and Q(1, 5) = 100. The result of the computation is Q(3, 1) = 80 because the reward is zero. The Q-table becomes:

$$Q=\begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 100 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 80 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

✓

Q-Learning

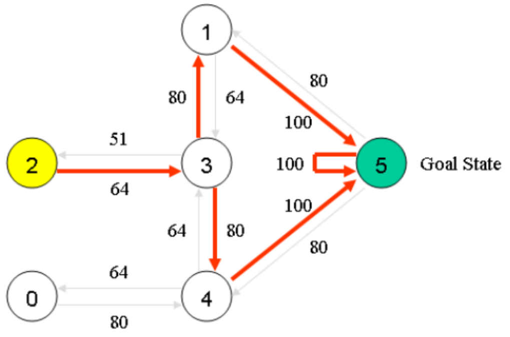
If our agent learns more through further episodes, it will finally reach convergence values in Q-table like:

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 400 & 0 \\ 1 & 0 & 0 & 0 & 320 & 0 & 500 \\ 2 & 0 & 0 & 0 & 320 & 0 & 0 \\ 3 & 0 & 400 & 256 & 0 & 400 & 0 \\ 4 & 320 & 0 & 0 & 320 & 0 & 500 \\ 5 & 0 & 400 & 0 & 0 & 400 & 500 \end{array}$$

This Q-table, can then be normalized (i.e.; converted to a percentage) by dividing all non-zero entries by the highest number (500 in this case):

$$Q = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & 0 & 0 & 0 & 0 & 80 & 0 \\ 1 & 0 & 0 & 0 & 64 & 0 & 100 \\ 2 & 0 & 0 & 0 & 64 & 0 & 0 \\ 3 & 0 & 80 & 51 & 0 & 80 & 0 \\ 4 & 64 & 0 & 0 & 64 & 0 & 100 \\ 5 & 0 & 80 & 0 & 0 & 80 & 100 \end{array}$$

Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state. Tracing the best sequences of states is as simple as following the links with the highest values at each state.

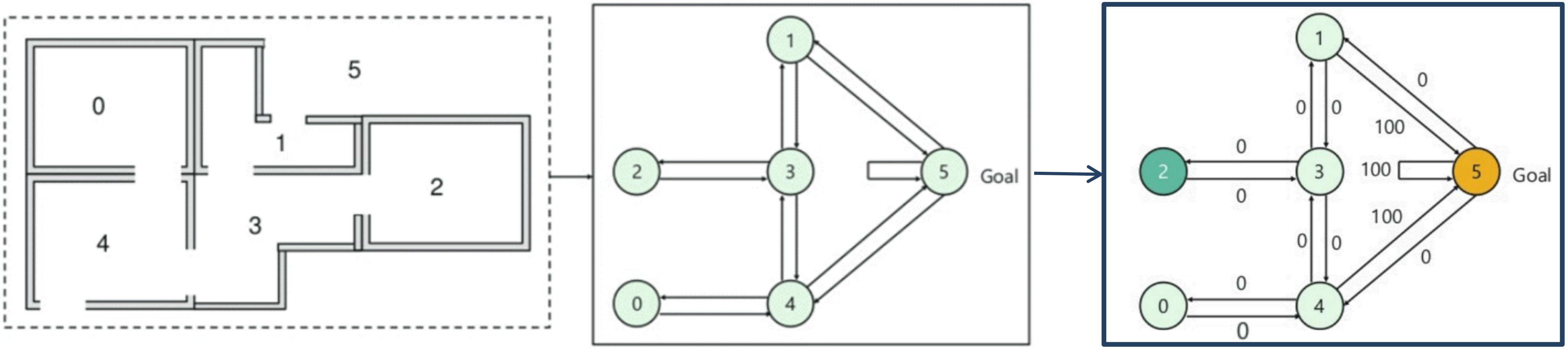


For example, from initial State 2, the agent can use the Q-table as a guide:
- From State 2 the maximum Q values suggest the action to go to state 3.
- From State 3 the maximum Q values suggest two alternatives: go to state 1 or 4. Suppose we arbitrarily choose to go to 1.
- From State 1 the maximum Q values suggest the action to go to state 5.
Thus the sequence is 2 – 3 – 1 – 5.

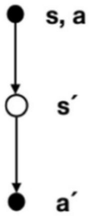
State-Action-Reward-State-Action (SARSA)

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

a_{t+1} s_{t+1}



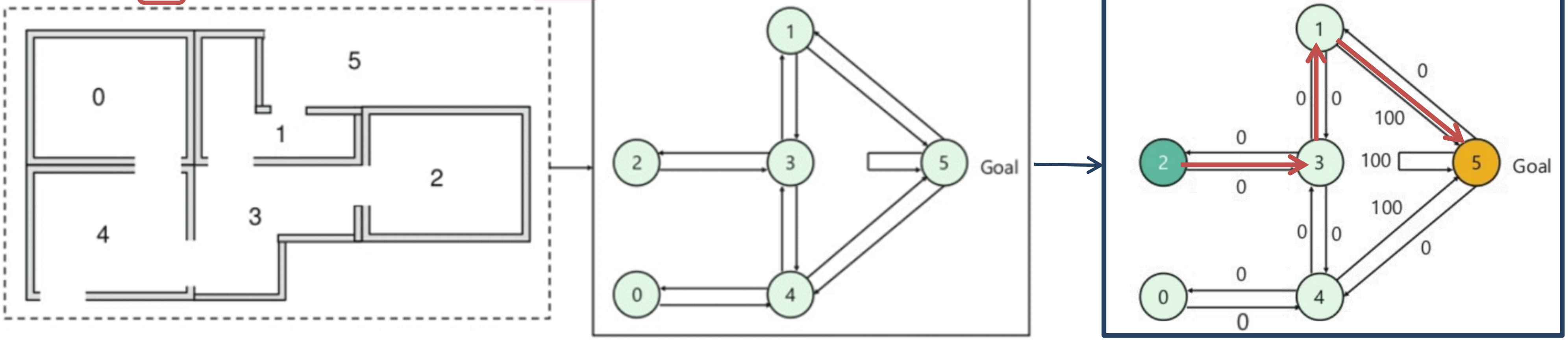
There is an ε-greedy for exploration in algorithm, meaning with ε probability, the agent will take action randomly. This method is used to increase the exploration because, without it, the agent may be stuck in a local optimal.



State-Action-Reward-State-Action (SARSA)

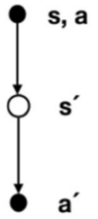
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, \boxed{a_{t+1}}) - Q(s_t, a_t)]$$

$\boxed{a_{t+1}}$ $\boxed{s_{t+1}}$



There is an ε-greedy for exploration in algorithm, meaning with ε probability, the agent will take action randomly. This method is used to increase the exploration because, without it, the agent may be stuck in a local optimal.

Path decided with random action selected

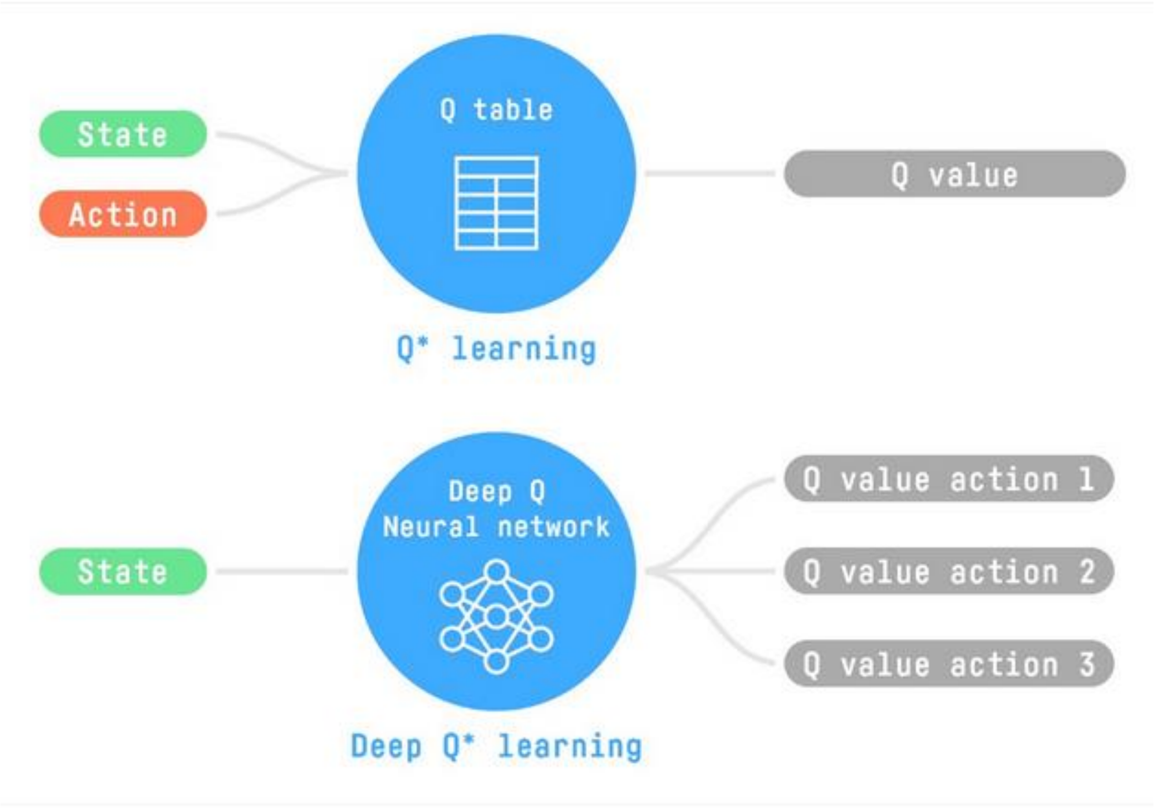


Demo – Inverted Pendulum

References

- Arthur L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers.” IBM Journal of Research and Development 3(3):210–229, 1959.
- Marvin Minsky, “Steps towards artificial intelligence.” Computers and Thought, 406–450, 1961.
- Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction”, The MIT Press, ISBN: 9780262039246, 2018.
- V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, “Playing Atari with Deep Reinforcement Learning”, Deepmind, 2013.
- Gustafsson, Robin, and Lucas Frojdendahl. Machine Learning for Traffic Control of Unmanned Mining Machines: Using the Q-learning and SARSA algorithms, 2019.

Deep Q-Learning



Schema inspired by the Q learning notebook by Udacity