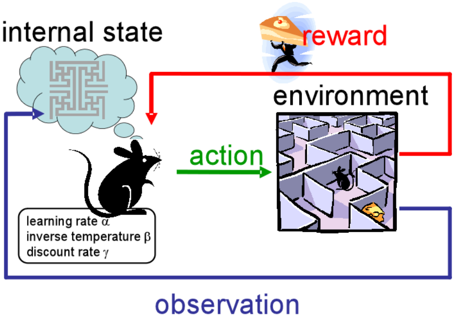
Basic Reinforcement Learning

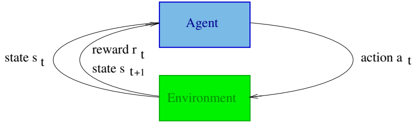
Reinforcement learning

Reinforcement Learning is about learning **behaviors**: taking **actions and decisions** in an automatic way in response to a **mutable external environment**.



Reinforcement learning problems

Problems involving an agent interacting with an environment, which provides numeric rewards



At time step t, the agent is in state s_t

- agent selects action a_t according to some policy $\pi(a_t|s_t)$
- environment answer with a local reward r_t
- and then enters into a new state s_{t+1}

A policy $\pi(a|s)$ is a probability distribution of actions given states.

Future Cumulative Reward

We want to learn the best way to act, that is, the best policy.

according to what objective?
we want to maximise the **future comulative reward**

Supposing to start at current time = 0,

$$R = r_1 + r_2 + r_3 \dots$$

or equivalently

$$R = \sum_{1 \leq i} r_i$$

Notation: big R for cumulative reward, small r for local rewards.

Future Cumulative Reward

We want to learn the best way to act, that is, the best policy.

according to what objective?
we want to maximise the **future comulative reward**

Supposing to start at current time $= 0$,

$$R = r_1 + r_2 + r_3 \dots$$

or equivalently

$$R = \sum_{1 \leq i} r_i$$

Notation: big R for cumulative reward, small r for local rewards.

Future Cumulative Reward

We want to learn the best way to act, that is, the best policy.

according to what objective?
we want to maximise the **future comulative reward**

Supposing to start at current time = 0,

$$R = r_1 + r_2 + r_3 \ldots$$

or equivalently

$$R = \sum_{1 \leq i} r_i$$

Notation: big R for cumulative reward, small r for local rewards.

Future Cumulative Reward

We want to learn the best way to act, that is, the best policy.

according to what objective?
we want to maximise the **future comulative reward**

Supposing to start at current time = 0,

$$R = r_1 + r_2 + r_3 \ldots$$

or equivalently

$$R = \sum_{1 \leq i} r_i$$

Notation: big R for cumulative reward, small r for local rewards.

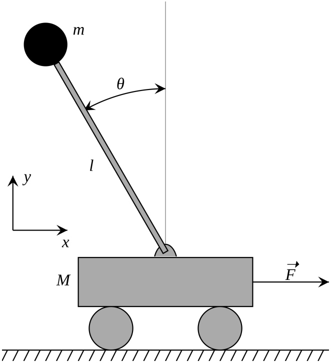
Future Discounted Cumulative Reward

We could also take into account the fact that distant rewards are less likely than close ones, that are more predictable.

To this aim we multiply the reward by a discount rate $0 < \gamma <= 1$ exponentially decreasing with time:

$$R = \sum_{1 \leq i} \gamma^i r_i$$

Example: Cart-pole problem

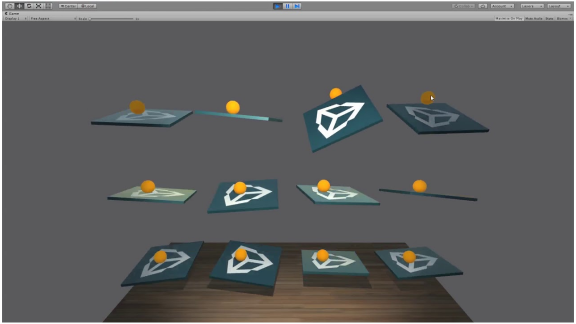


Objective: balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: +1 at each time step if the pole is upright

3D balancing

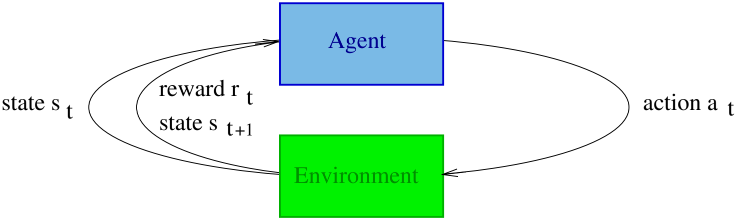
A similar problem in 3D, using the Unity simulation framework (the "ultimate game development platform").



Video on you tube.

Mathematical formulation

How can we formalize the RL problem?



Markov Decision Process

Markov property: Current state completely characterises the state of the world: future actions only depend on the current state.

Defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$
$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: reward probability given (state, action) pair
$\mathcal{P}$: transition probability to next state given (state, action) pair
γ: discount factor

The optimal policy

At time step $t = 0$, the environment in state s_0.
Then, for $t = 0$ until done:

- agent selects action a_t according to some policy $\pi(a_t|s_t)$
- environment samples reward $r_t \sim R(r_t|s_t, a_t)$
- environment samples next state $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$

A policy produces trajectories (or paths) $s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots$

We want to find an optimal policy, that is

$$\pi^* = argmax_\pi \, \mathbb{E} \sum_{t \geq 0} \gamma^t r_t$$

where the average is taken over all possible trajectories.

Model-free vs. model-based

The transition from state s_t to state s_{t+1} is not always determinstic, but governed by some probability $P(s_{t+1}|s_t, a_t)$.

If the learning model needs to learn this probability $P(s_{t+1}|s_t, a_t)$, then it is called model-based.

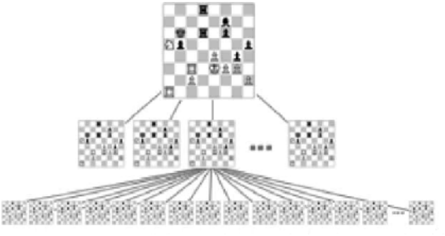
In model-free approaches, this information is left implicit: you learn to take actions from past experience relying on trial-and-error.

We shall mostly investigate **model free** approaches.

Model-based, planning, simulation

Having knowledge of the model (transitions and rewards) we can build a **simulator**.

Having a simulator, we can do **planning** (e.g. explore trajectories up to a certain horizon and select the one with maximum return)



But building a simulator is a **complex** task, and in some cases an **impossible** one.

Model-free approaches

Do we really need knoweldge of the model to decide the best action?

When we drive, do we have a precise comprehension of the effect that our actions will have on the dymamics of the car?

Frequently, we are not even aware of the current speed of the car.

What we learn (in an almost unconscious way) is that given the road ahead we need to re(act) in given way.

Model-free approaches

Do we really need knoweldge of the model to decide the best action?

When we drive, do we have a precise comprehension of the effect that our actions will have on the dymamics of the car?

Frequently, we are not even aware of the current speed of the car.

What we learn (in an almost unconscious way) is that given the road ahead we need to re(act) in given way.

Model-free approaches

Do we really need knoweldge of the model to decide the best action?

When we drive, do we have a precise comprehension of the effect that our actions will have on the dymamics of the car?

Frequently, we are not even aware of the current speed of the car.

What we learn (in an almost unconscious way) is that given the road ahead we need to re(act) in given way.

Exploration/Exploitation trade off

Reinforcement learning requires acquisition of experience interacting with the environment.

All techniques have to deal with the exploration/exploitation trade-off.

Exploration is finding more information about the environment, usually requiring randomicity

Exploitation is taking advantage of the available information to direct and possibly improve the exploration

Model free approaches

Two basic techniques:

Value-based We try to evaluate color orange each state s with a value function $V(s)$. The policy is implicit: we shall choose the action taking us to the next state with the best evaluation.

Policy-Based we directly try to improve the current policy, hopefully optimizing it. Remember that the policy defines the agent behavior at a given state:

$$a = \pi(s)$$

better, $\pi(s)$ is the probability to perform a in state s.

Next arguments

Value-based approaches

Value function and Q-function

Let us assume a given policy π.
How good is a state?

$$V(s) = \mathbb{E}_{s_0=s} \sum_{t \geq 0} \gamma^t r_t$$

How good is action a for state s?

$$Q(s,a) = \mathbb{E}_{\substack{s_0=s \\ a_0=a}} \sum_{t \geq 0} \gamma^t r_t$$

Expectations are on all trajectories defined by the given strategy.

Relation between V and Q

We can easily compute V from Q

$$V(s) = \sum_a \pi(a|s) * Q(s,a)$$

i.e. we sum every action-value weighted by the probability $\pi(a|s)$ to take that action.

However, to compute Q from V we would need (model-based!!) knowledge of the state s' we are likely to end up by taking action a:

$$Q(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) * V(s')$$

Optimal policy

The optimal Q-value function $Q^*(s,a)$ is the maximum expected cumulative reward achievable from state s performing action a:

$$Q^*(s,a) = max_\pi \mathop{\mathbb{E}}_{\substack{s_0=s \\ a_0=a}} \sum_{t\geq 0} \gamma^t r_t$$

The optimal policy π^* consists in taking the best action in any state as specified by Q^*

Bellman equation

The Bellman equation expresses a relation between the solution for a given problem in terms of the solutions for subproblems.

Q^* satisfies the following Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'}[r_0 + \gamma max_{a'} Q^*(s',a')]$$

Indeed, $R_{s'} = max_{a'} Q^*(s',a') = V^*(s')$ is the optimal future cumulative reward from s', and the optimal future cumulative reward from s when taking action a is $r_0 + \gamma R_{s'}$

The optimal policy π^* consists in taking the best action in any state as specified by Q^*

Computing Q^* via iterative update

We know that Q^* satisfies the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s'}[r_0 + \gamma max_{a'} Q^*(s',a')]$$

The idea is to use it to perform **iterative update** on progressive approximations Q^i of Q^*:

$$\underbrace{Q^{i+1}(s,a)}_{\text{next estimation}} \leftarrow \underbrace{Q^i(s,a)}_{\text{current estimation}} + \alpha(\underbrace{r_0 + \gamma max_{a'} Q^i(s',a') - Q^i(s,a)}_{\text{recursive update}})$$

α is a learning rate.

The recursive update is the derivative of the quadratic distance between $Q^i(s,a)$ and $r_0 + \gamma max_{a'} Q^i(s',a')$ that should be equal, according to the Bellman equation.

Q-learning pseudo code (first approx)

0. initialize the Q-table
1. repeat until termination of the episode:
2. choose action a in current state s
   according to the current Q-table
3. perform action a and observe reward r and new state s'
4. update the table:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_0 + \gamma max_{a'} Q(s',a') - Q(s,a))$$

Q-learning transitions

$$\underbrace{Q^{i+1}(s,a)}_{\text{next estimation}} \leftarrow \underbrace{Q^{i}(s,a)}_{\text{current estimation}} + \alpha\big(\underbrace{r_0 + \gamma max_{a'} Q^{i}(s',a') - Q^{i}(s,a)}_{\text{recursive update}}\big)$$

In order to perform an update, all the information we need is contained in a tuple (transition):

$$(s, a, r, T, s')$$

where:

- s is the current state
- a is the action done
- r is the reward obtained
- T is a boolean stating the termination of the episode
- s' is the new state after doing the action

Q-learning transitions

Transitions (s,a,r,T,s') are collected by exploring the environment.

Each tuple is independent from the others.

They can be saved into an experience replay buffer and re-executed at leisure. Better than learning from batches of consecutive samples because:

- consecutive samples are correlated $\Rightarrow$ inefficient learning
- great risk of introducing biases during learning by exploiting unbalanced sets of transitions

Q-learning is an off-policy techique: it does not rely on any policy, and only needs local transitions (tuples).

Exploration vs. exploitation

At start, the Q-table is **not informative**.

Taking actions according to it could introduce biases, and **prevent exploration**.

In early stages, we want to privilege random exploration, and start relying more on the table when more experience is acquired.

epsilon greedy strategy

We specify an exploration rate ϵ, initially equal to 1.

This is the rate of steps that done randomly.

We generate a random number. If this number is larger then ϵ, then we choose the action according to the information collected in the Q-table (exploitation); otherwise we choose the action at random (exploration)

We progressively reduce ϵ along training.

epsilon greedy strategy

Q-learning pseudo code (revisited)

initialize the Q-table, Replay Buffer D, $\epsilon = 1$
repeat for the desired number of episodes:
  initialize state s
  repeat until termination of the episode:
    with probability ϵ choose a random move a
    otherwise $a = max_a Q(s, a)$
    perform action a and observe reward r and new state s'
    store transition (s, a, r, T, s') in D
    sample random minibatch of transitions (s, a, r, T, s') from D
    for each transition in the minibatch:

$$R = \begin{cases} r & \text{if } T \\ r + \gamma max_{a'} Q(s', a') & \text{if not } T \end{cases}$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R - Q(s, a))$$

    decrement ϵ

Next arguments

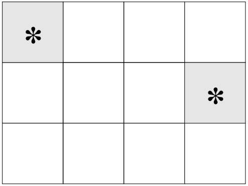
Example

A simple MDP: Grid World

Actions = {

1. right
2. up
3. left
4. down

}



A negative reward for each transition (e.g. r = -1)

Objective: reach an exit (greyed out) in least number of steps

About the Grid World

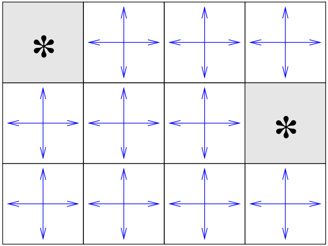
The grid world is an **abstraction**. Each cell is a different state and we can pass from a state to another taking actions.

*			
			*

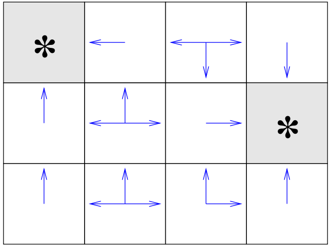
The visibility of the agent is **confined to its current cell**.
He can only choose an action; it is the action that determines the new state we end up into.

We will see that little by little, by **trial and error**, the agent will discover the good way of action that will allow him to reach a winning position starting from any state.

Policies for the Grid World

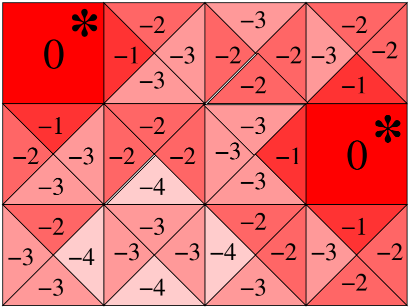


random policy



optimal policy

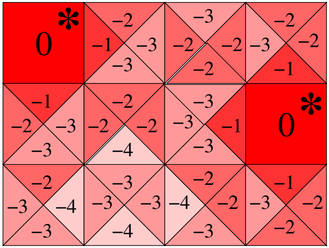
Optimal Q-value $Q^*(s, a)$



If $s \xrightarrow{a} s'$, then (Bellman's equation)

$$Q^*(s, a) = r + max_{a'} Q^*(s', a') = -1 + V^*(s')$$

Q-value and V-value



Optimal Q-value

0 *	−1	−2	−1
−1	−2	−1	0 *
−2	−3	−2	−1

Optimal V-value

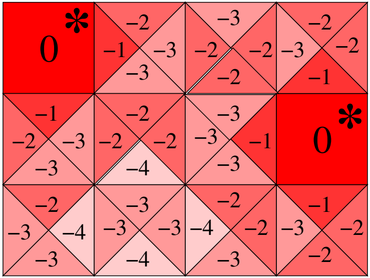
The V-value is just the max of the Q-values, over all possible actions:

$$V(s) = max_a Q(s, a)$$

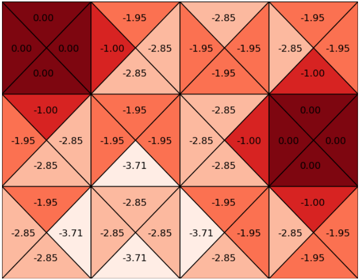
Demo

```
for n in range(0,episodes):
  s0 = random_state()
  while not term(s0):
      #choose action
      if np.random.random() > epsilon:
          a = np.argmax(Qtable[s0]) #exploit Qtable
      else:
          a = np.random.randint(4) #random move
      s1 = move(s0,a)
      T = term(s1)
      if T:
          R = -1
      else:
          R = -1 + gamma*np.max(Qtable[s1])
      Qtable[s0][a] = Qtable[s0][a] + alpha*(R-Qtable[s0][a])
      s0 = s1
  epsilon = epsilon * epsilon_rate #decrease epsilon
```

Results



Theoretical optimal Q-value



Result of the algorithm

Result of algorithm after 10000 iterations.

How learning works in practice

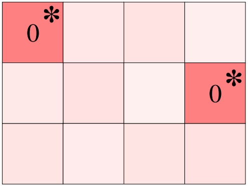
When we start, we only know the right values for terminal states.

The other states will get a random value.

Actions = {

1. right
2. up
3. left
4. down

}



A negative reward for each transition (e.g. r = -1)

How learning works in practice

Most of the actions produce meaningless updates, since the current estimation of the Q-value function is erroneous

The relevant actions are those leading to states whose Q-value is accurate; at the beginning these are just terminal states