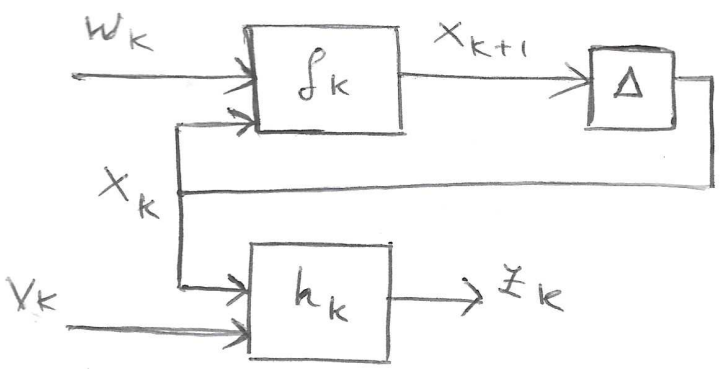
16 Markov Decision Processes and Reinforcement Learning

RL is the ability of an **agent** to learn a behaviour, up to now we did not consider situation in which something change over time. We learn a function that depends also on time.

The classical view of a dynamic system:



Dynamic system: STATE EVOLVES OVER TIME and the state is the representation of the information.

The state is denoted with x.

w, v : noise
f: state transition model
h: observation model
z: observations.
x: state

The evolution is given by a SEQUENCE of STATE. The sys. evolves because of a dynamic transition function f_k, that is responsible to model how state evolve over time.

The EVOLUTION IS AFFECTED BY THE NOISE, that is what it is not modeled in the state (i.e. wind for a humanoid robot).

STATE → observable
STATE → not observable

We usually have an observation model h_k able to EXTRACT INFORMATION FROM THE STATE

TASK: We want that our agent can drive the evolution of the state: CONTROL THE SEQUENCE OF STATES TO REACH SOME GOAL.

Two possible way of doing:

1 **REASONING**: if we assume to know f and h, given the model (f, h) and the current state x_k, we can predict the future (x_{k+T}, z_{k+T}).
techiques.

I CAN ESTIMATE THE FUTURE STATE by applying f. I don't know to execute actions, Just predict.

2 **LEARNING** alternative

I don't know how the sys'. works, I cannot predict the future, but I can try. Given past experience $(z_{0:k})$, determine the ~~x~~model (f, h).

EX: PLAY CHESS MANY TIMES, until the agent can win the game

The state (ENCODE SNAPSHOT at time t) →
- all the past knowledge needed to predict the future
- the knowledge gathered through operation
- the knowledge needed to pursue the goal.

Configuration of a board game, screenshot of a video game, ...

When the state is fully observable, the decision making problem for an agent is TO DECIDE WHICH ACTION MUST BE EXECUTED IN A GIVEN STATE.

The agent should compute a function to perform decision in a given state.

$\pi : X \longmapsto A$ ← POLICY or BEHAVIOUR FUNCTION

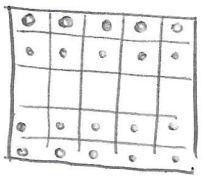
Maps state to action

$\forall$ state x, the agent should know which is the action to be executed.

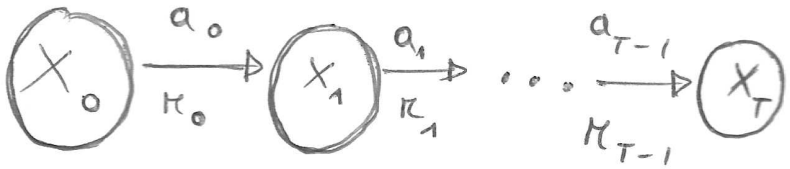
learning a function is a typical goal of standard supervised learning. Suppose that I have a set of discrete actions ($\uparrow, \downarrow, \leftarrow, \rightarrow$ in a video game), deciding for the best action can be seen as an instance ~~of~~ a classification problem, but I need a dataset of pairs input-output like this x_i, a_i. I should need someone that tells which is the best action for a given state. IN GENERAL, WE ASSUME TO NOT HAVE A DATASET LIKE THIS (since it is very difficult to obtain).

In RL the dataset is given by a sequence of episodes. Suppose to have a dynamic system, where we can well define duration of episodes, such as chess, then is an initial configuration of the board and there are well defined end-configurations.

CHESS-BOARD



Any chess game is a temporal evolution of states starting from the same initial config.



We measure HOW GOOD AN ACTION IS BY THE CONCEPT OF REWARD (a real value)

$$D = \left\{ \langle x_1, a_1, r_1, \dots, x_M, a_M, r_M \rangle_{i=0}^{(i)\,N} \right\}$$

not the best action, is just an ACTION

I have many sequences, in chess I have set of games in D.

Typical exam question: "What is the difference between supervised learning an RL?"

ANSWER: Both learns a function, the real difference is in the dataset; in supervised learning I have pairs of input - output of the function, while in RL I have a dataset in which for each evolution of the system I collect rewards.

How assign rewards? This methods works pretty well even if WE USE A VERY SIMPLE WAY OF ASSIGNING REWARDS. For instance in case of chess:

$$r = \begin{cases} r_i = 0 \quad \forall i \in \{0, M-2\} & \text{intermediate moves.} \\ r_{M-1} \begin{cases} 100 & \text{won} \\ 0 & \text{drawn} \\ -100 & \text{lost} \end{cases} & \text{For the last move} \end{cases}$$

In RL ALGORITHMS ARE DOMAIN INDEPENDENT, i.e. to learn to play chess I do not to encode the rule of chess, sequence of actions, states and rewards is what matters.

The dynamic system representation we consider can be:

X: STATES
- explicit discrete and finite repr. $X = \{x_1 \dots x_n\}$
- continuos representation $X = F(\,)$
- probabilist representation $P(X)$

A: ACTIONS
- explicit discrete and finite repr. $A = \{a_1 \dots a_m\}$
- continuos representation $A = U(\dots)$ (control function)

δ : TRANSITION FUNCTION : deterministic / non-det. / probabilistic.

Z : SET OF OBSERVATION : same for the set of states.

We will consider only on some aspects on the representation:

Finite repr. for states ⊕ Finite repr. for actions ⊕ probabilistic δ ⊕ NO OBSERV. Z since we assume that the sys is FULLY OBSERV.

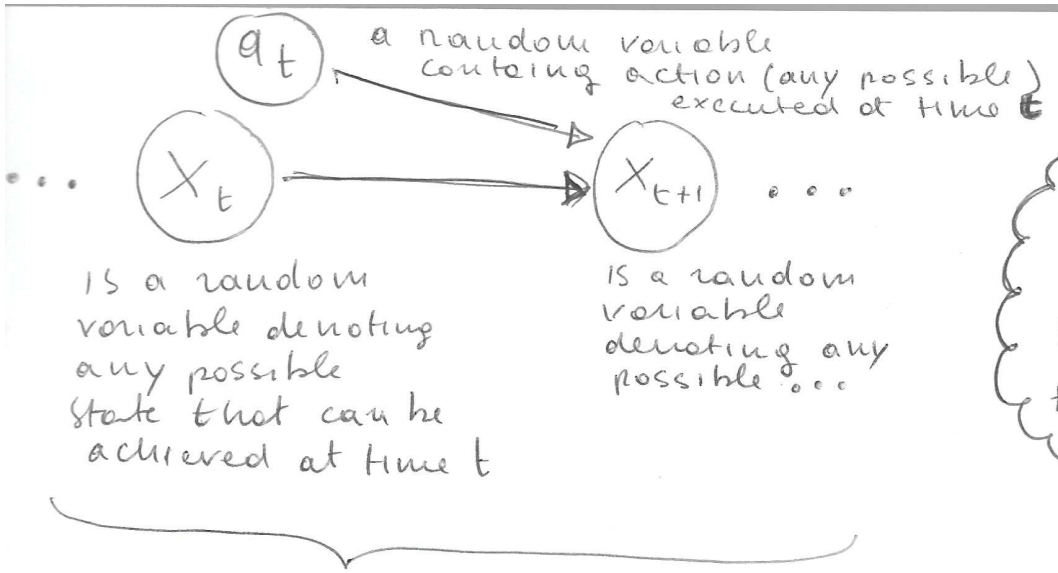
MARKOV PROPERTY (very important, since allow to reduce complexity).

The knowledge of the current state is all we need in order TO MAKE DECISION FOR THE FUTURE, we don't need the history (the past).

- Once the current state is known, the evolution of the dynamic system does <u>not depend</u> on the history of states, actions and observations.
- THE CURRENT STATE CONTAINS ALL INFORMATION NEEDED TO PREDICT THE FUTURE.
- Future states are conditionally independent of past states and past observations given the current state
- THE KNOWLEDGE OF CURRENT STATE MAKES PAST, PRESENT and FUTURE OBSERVATIONS STATISTICALLY INDEPENDENT.

<u>Markov process</u> is a process that has the Markov property.

This an important semplification, drop many dependencies. In chess Markov property holds.



X_t is a random variable denoting any possible state that can be achieved at time t

X_{t+1} is a random variable denoting any possible ...

Once I know X_t all the previous information are not needed to predict X_{t+1}

Bayes network

X_{t+1} has no other arrow because of Markov assumption

In general:

$$P(X_{t+1} \mid X_t, X_{t-1}, \dots, X_0) \xrightarrow{\text{Markov ass.}} \boxed{P(X_{t+1} \mid X_t)}$$

NOT TRUE IN GENERAL

Markov chain: sequence of random variables having Markov property. It is valid also without actions.

With actions:

$$P(X_{t+1} \mid X_t, A_t, \dots) \Rightarrow P(X_{t+1} \mid X_t, A_t)$$

STATES ARE FULLY OBSERVABLE: the agent is able to fully understand the current state, it does not mean that it can predict the next state. In presence of non-deterministic or stochastic actions, the state resulting from the execution of an action is not know before the execution of the action, but it can be fully observed after its execution.

MDP $= \langle X, A, \delta, r \rangle$ — transition function $\delta : X \times A \mapsto X$

Markov decision process

X: set of states

A: set of action

r: reward function $r : X \times A \to \mathbb{R}$

They may have different formulation according to the MDP considered. Here we consider the deterministic version of the transition function and the reward function.

NON DETERMINISTIC

When an agent execute an action from a state, the outcome is not known. In this case the δ returns a set of possible states!

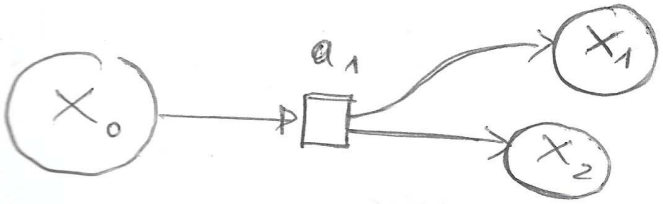
$$\delta_{NA} : X \times A \mapsto 2^X$$

Once the execution of the action is completed we know which is the new state. The reward function is also extended, it is defined in terms of current state, current action and next state:

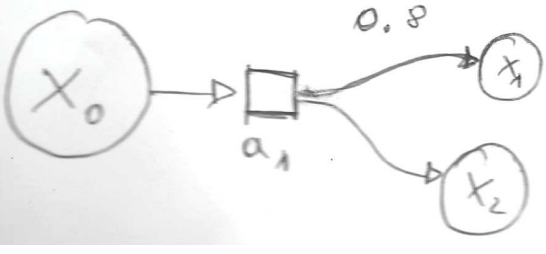
$$r : X \times A \times X \mapsto \mathbb{R}$$

The most general formulation is probabilistic or stochastic. Given a state x and an action a we have a prob. distrub. to reach another state (extension of non det. case)

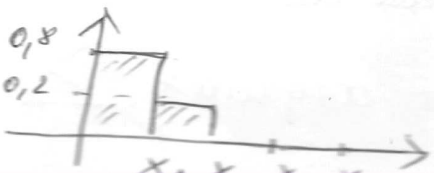
$$\delta_{ST} : P(x' \mid x, a)$$



NON DETERMINISTIC CASE: $\delta(x_0, a_1) = \{x_1, x_2\}$

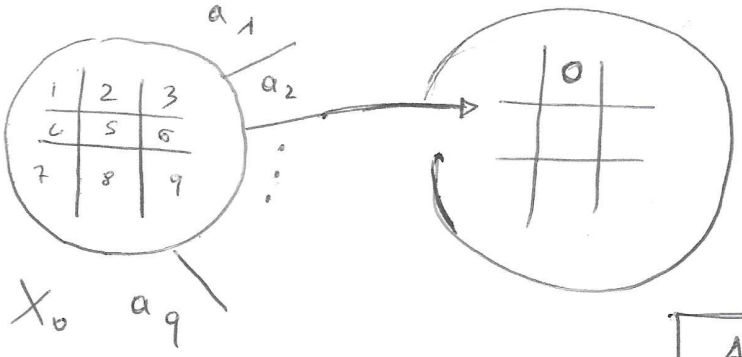


STOCHASTIC CASE



Note: Chess game is deterministic, the MDP associated is non-deterministic because there is the opponent.

Tic-ta-toe



This is not a state of the MDP, it is now the opponent turn and he/she chooses a move in a non deterministic way.

A DETERMINISTIC EVOL. OF THE SYST. MAKES A NON DETERMINISTIC DEFINITION OF THE PROBLEM

I want to learn the policy function

$$\pi : X \mapsto A$$

I WANT TO LEARN THE BEST POLICY

For each state $x \in X$, $\pi(x) \in A$ is the optimal action to be executed in such state.

We need to define the optimality on the rewards. Given a policy in deterministic case, it will produce a sequence of state that is deterministic, during this evolution I can collect rewards. In general WE WANT TO MAXIMIZE THE REWARDS; FOR AN AGENT IS BETTER TO HAVE THE REWARD SOONER, so usually you consider a discount factor that penalizes rewards that are in the future.

$$\sum_{i=0}^{T} \gamma^i r_i = r_0 + \gamma r_1 + \gamma^2 r_2 \dots \qquad \gamma \in [0,1]$$

DISCOUNTED CUMULATIVE REWARD

We compute the policy that maximizes this quantity

$$\pi^* = \operatorname{argmax}_{\pi} \sum \gamma^i r_i$$

In non deterministic execution, I have to consider the expected value!

$$V^{\pi}(x) \equiv E\left[\sum_{i=0}^{T} \gamma^i r_i\right]$$

I HAVE TO AVG THE POSSIBLE INFINITE NUMBER OF EXPERIMENT.

This is also what happens in the stochastic case. This function is called the VALUE FUNCTION OF A POLICY FOR A GIVEN STATE (How good is for the sys to execute that policy).

π^* IS AN OPTIMAL POLICY iff for any other policy π:

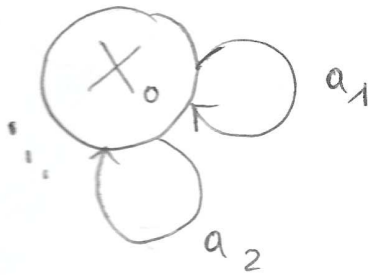
$$V^{\pi^*}(x) \geq V^{\pi}(x), \forall x.$$

{ If every state can be the initial state, we have to repeat for each state! }

For **infinite horizon** problems, a stationary **MDP** always has an optimal policy.

infinite horizon $\Rightarrow$ the agent does not know if the action that is going to execute is the last (does not know the lenght of the episode).

One-State Markov Decision Process



- x_0 unique state
- A finite set of actions
- $\delta(x_0, a_i) = x_0 \quad \forall a_i \in A$
- $r(x_0, a_i, x_0) = r(a_i)$

$MDP = \langle \{x_0\}, A, \delta, r \rangle$

$\pi^*(x_0) = a_i$

Let's consider the possible cases of this model:

- δ, r ARE DETERMINISTIC and you know in advance the reward function. The optimal policy:

$$\pi^*(x_0) = \underset{a_i \in A}{\operatorname{argmax}}\, r(a_i)$$

JUST TAKING THE ACTION MAXIM.

- the reward are deterministic but they are not known.

|A| iterat. are needed

ALGORITHM

$\forall a_i \in A$:

execute a_i and collect κ_i

optimal policy : $\pi^*(x_0) = a_i$, with $i = \text{argmax } \kappa_i$

- the reward is NON-DETERMINISTIC and known. You know a probability distribution, i.e. a Gaussian distribution for each κ_i.

OPT : $\pi^*(x_0) = \underset{a_i \in A}{\text{argmax}} \left(E[\kappa(a_i)] \right)$ → maximize the mean of distribution.

- the reward is non deterministic and unknown. Here you have to try, but not once because the outcome is non-det.

We need some data structures that contains informations. We have to repeat a set of experiments. Let's see the main structure of the algorithm:

1. Initialize a data structure θ
2. For each $t = 1 \dots T$ ← $T >> |A|$ MANY ITERATIONS ARE NEEDED
   - 2.1 choose an action $a_t \in A$
   - 2.2 execute a_t and collect κ_t
   - 2.3 update θ
3. $\pi^*(x_0) = \dots$ according to the data structure θ

Let's see an instantiation of this algorithm where θ is a vector. I use also a counter c used to compute the rewards.

1. Initialize $\theta_{(0)}[i] \leftarrow 0$ and $c[i] \leftarrow 0$, $i = 1 \dots |A|$
2. $\forall\ t = 1, \dots, T$
   - 2.1 choose an index $\hat{i}$ for action $a_{(t)} = a_{\hat{i}} \in A$
   - 2.2. execute a_t and collect r_t
   - 2.3. increment $c[\hat{i}]$
   - 2.4. update $\theta_t[\hat{i}] = \frac{1}{c[\hat{i}]} \left(r_{(t)} + \left(c[\hat{i}] - 1 \right) \theta_{(t-1)}[\hat{i}] \right)$
3. Optimal policy $\pi^*(x_0) = a_i$, with $i = \underset{i=1..|A|}{\operatorname{argmax}} \theta_T[i]$

At each step we update our datastructure and when you choose an action a_i you affect only the component i.

Let's extend this to more than one state, you keeps the same idea of the algorithm and you have to deal with evolution of the state.

Given an agent accomplishing a task according to an MDP $\langle X, A, \delta, r \rangle$ for which functions δ and r are UNKNOWN to the agent, the task is to learn the best policy. Since δ and r are unknown the agent can't predict the effect of its actions, BUT IT CAN EXECUTE THEM and THEN OBSERVE THE OUT-COME. The learning task is thus performed by repeating these steps!

- CHOOSE an ACTION a
- EXECUTE a
- OBSERVE the new state
- COLLECT the reward.

In order to compute the optimal policy there are two family of approaches.

(1) VALUE ITERATION (estimate the Value function and then compute π)

(2) POLICY ITERATION (estimate directly π)

Value iteration

If we can estimate the value function the optimal policy can be estimated easily:

$$\pi^*(x) = \underset{a \in A}{\arg\max} \left[r(x,a) + \gamma \underbrace{V^*(\delta(x,a))}_{\text{the best that I can get in the future}} \right]$$

But this policy cannot be computed because δ and r are not known. Let's define a new function **Q Function**, also called state-action function, because is defined on a pair (x, a).

$$Q^{\pi}(x,a) = \underbrace{r(x,a)}_{\text{immediate reward}} + \gamma \underbrace{V^{\pi}(\delta(x,a))}_{\text{The value of the policy from the next state}} \qquad \text{DETERMIN.}$$

$$Q^{\pi}(x,a) = \sum_{x'} \underbrace{P(x' \mid x,a)\left(r(x,a,x') + \gamma V^{\pi}(x')\right)}_{\text{weighted avg (probabilities are the weights) of the possible successor state } x'} \qquad \text{NON-DETER.}$$

Once we have Q we can easily compute the optimal policy;

$$\boxed{\pi^*(x) = \underset{a \in A}{\arg\max}\, Q(x,a)}$$

TRAINING RULE TO LEARN Q:

Deterministic case:

$$Q(x_t, a_t) = r(x_t, a_t) + \gamma \underbrace{\max_{a' \in A} Q(x_{t+1}, a')}_{\text{Max of all possible actions that can be executed from a state correspond to the value function of that state.}}$$

$\hat{Q}$ approximation of Q

$$\hat{Q}(x,a) \leftarrow \underbrace{r}_{\text{immediate reward}} + \gamma \max_{a'} \hat{Q}(x', a')$$

(x' is the state resulting by applying a in state x)

Q Learning algorithm → we consider also the evolution of the state.

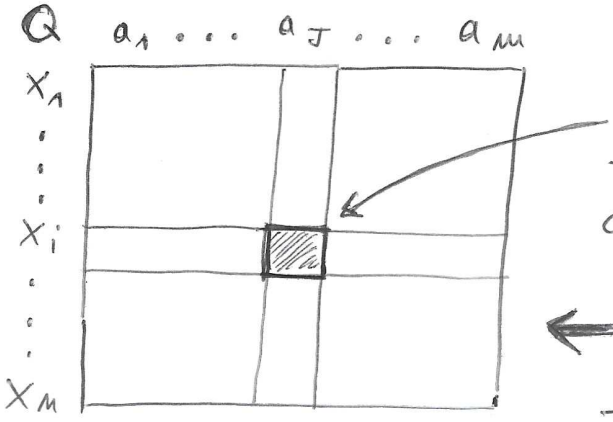
1. for each x, a initialize table entry $\hat{Q}_0(x,a) \leftarrow 0$
2. observe current state x
3. $\forall \; t = 1 \dots T$

REPEAT MANY TIMES:

- 3.1 choose an action a
- 3.2 execute the action a
- 3.3 observe x'
- 3.4 collect $\overline{r}$
- 3.5 update $\hat{Q}_{(t)}(x,a) \leftarrow \overline{r} + \gamma \max_{a' \in A} \hat{Q}_{(t-1)}(x', a')$
- 3.6 $x \leftarrow x'$

4. OPT $\pi^*(x) = \text{argmax}_{a \in A} Q_{(T)}(x,a)$

Since both X and A are discrete the Q function can be described in a tabular way:



the one that is updated by the algorithm. This the update for the deterministic case.

← THIS REPRESENT. CAN BE FEASIBLE ONLY IN THE CASE OF A SIMPLE DOMAIN. WE NEED TO REPRESENT THE INFO IN A DIFFERENT WAY.

We need to still understand how to choose an action!

Note: the Dataset in RL is built online during the execution of the algorithm, you don't even need to store the data. This is enforced by the Markov property, forget about the past.

How actions are chosen by the agents?

(I) EXPLOITATION: select an action a that maximizes $\hat{Q}(x,a)$

(II) EXPLORATION: select action a with low value of $\hat{Q}(x,a)$

reach the goal faster (I) **VS** check for (II) better opportunities

TRADE-OFF

ε - greedy strategy

Given $0 \leq \varepsilon \leq 1$ select a random action with probability ε, select the best action with probability $1-\varepsilon$.

ε value can decrease overtime (first exploration, then exploitation)

we cannot avoid to do both exploitation and exploration, because if we do only exploration we may reach the goal slowly

Soft max strategy

actions with higher $\hat{Q}$ values are assigned higher probabilities, but every action is assigned a non zero probability.

$$P(a_i|x) = \frac{k^{\hat{Q}(x,a)}}{\sum_j k^{\hat{Q}(x,a_j)}}$$

$k > 0$ how strongly the selection favors actions with high $\hat{Q}$ values.

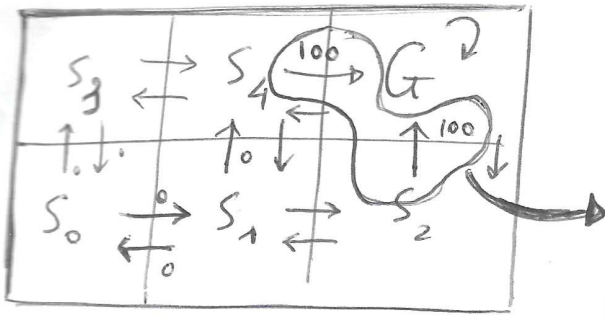
Example of Q-Learning in the deterministic case.

Let's consider [redacted] one initial state, we don't know anything of the dynamic of the system (δ, r are UNKNOWN). We can RECOGNIZE WHEN THE GOAL IS REACHED (our goal-state).

- Let's consider the case when the reward is 0 for each action, except for the one that take the agent in the goal-state ($r = 100$).

- If I have failure state I can add negative reward.

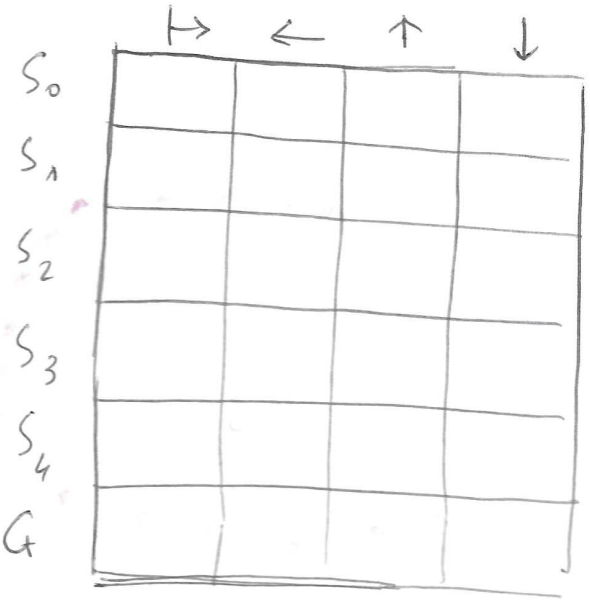
Environment of 6 positions and an agents that can move in the cardinal directions:



We are not able to predict the result, but we can observe what happens.

ONLY THIS TWO WILL HAVE REWARD 100; (the agent does not know R function).

We apply the q-learning by using $\gamma = 0.9$. Using $\gamma < 1$ means that the agent want to reach the goal as soon as possible. Let's represent the Q table:



- THIS matrix is initialized to Zero.
- We repeat a certain number of time a set of steps (choose a, execute a, ...) and update Q-table.
- Note: if some actions are not avaiable, you remain in that state (such as "←" in S_3).
- UPDATE RULE: $Q(x,a) \leftarrow R + \gamma \underbrace{\max_{a' \in A} \hat{Q}(x',a')}_{\text{the best that I can do in the future.}}$

Let's consider that the agent will choose an action at random (uniform distribution).
Any movement of the agent will not update the Q-table, because the reward is zero for all intermediate actions. The update there is only when the agent reach the goal, for instance at some point the agent is in S_2 and decides to go ↑.

$$Q(S_2, \uparrow) \leftarrow 100$$

There are 2 situation in which the table can change!

- S_4 , $\rightarrow$ (similar to S_2 , $\uparrow$)
- $S_1 \rightarrow S_2$ because $r = 0$ but the successor state is S_2 and the maximum value is 100.

Q

	→	←	↑	↓
S_0	81			
S_1	90			
S_2			100	
S_3				
S_4				81
G				

Then I can continue to update the values.

When I STOP the process I will have an estimation of the Q function from which I can compute the optimal policy;

Suppose that we stop with this Q!

	→	←	↑	↓
S_0			63	
S_1	90			
S_2			100	
S_3	72			
S_4				81
G				

$\Rightarrow$ π^*

↑	S_0
→	S_1
↑	S_2
→	S_3
↓	S_4
-	G

This is not the best action for every state, since the agent did not try to go directly to G from S_4.

IN PRINCIPLE YOU HAVE TO TRY ALL ACTION-STATE PAIR.

Note: In some real-world application you cannot start from whatever state.

IF YOU VISIT ALL STATE ACTION PAIRS YOU WILL GET THE OPT POLICY, but it is not feasible! In the non deterministic case you should visit every pair infinitely often, but this is not possible!

What we will get is an approximation of the optimal policy.

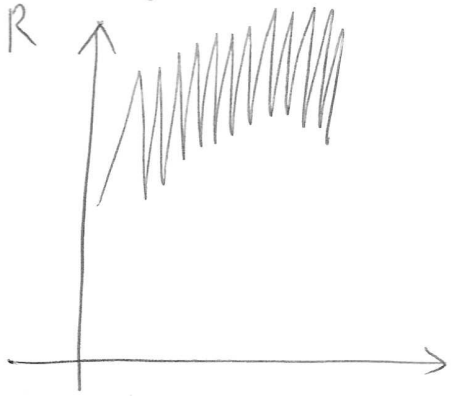
In the deterministic case values in the Q-table never change, and the values converge to the optimal case. ONCE WE HAVE THE Q-TABLE WE CAN COMPUTE THE OPTIMAL POLICY.

EVALUATION METHODS

In many practical case how we can evaluate reinforcement learning? The typical evaluation metric is the reward function over time, we compute how the cumulative reward function evolves overtime.

The curve in general is not smooth, because there are two main source of noise:

In many real cases the reward function is like this



1. EXPLORATION
At some point the agent want to explore and increase performance in the future.

2. NON DET - EFFECT OF THE ENVIRONMENT.

What is important is to remove the noises in the evaluation of the system. During the learning process, we repeat until termination condition!

1. Execute K steps of learning
2. Evaluate the current policy π_K, we stop exploration and use the best policy computed so far ⟶ we REMOVE EXPLORATION NOISE.

In the non deterministic case we can make an average of the result obtained.

10. Reinf. Learning - Non-deterministic

In non deterministic case when an action is executed we have a probability distribution over all possible state, also the reward changes because the reward of an action depends on the successor state.

$$MDP = \langle X, A, \delta, r \rangle$$

X = finite set of states

A = finite set of actions

$P(x'|x,a')$ is a prob. distribution over transition

$r(x,a,x')$ is a reward function.

HOW WE EXTEND THE CONCEPT OF THE VALUE FUNCTION and Q FUNCTION ?

$$V^{\pi}(x) = E\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]$$

$$\pi^* = \arg\max_{\pi} V^{\pi}(x) \quad \forall x$$

let's see the Q function!

$$Q(x,a) = E[r(x,a)] + \gamma \sum_{x'} P(x'|x,a) \max_{a'} Q(x',a)$$

$$\pi^* = \arg\max_{a \in A} Q^*(x,a)$$

BEST I CAN DO IN THE FUTURE: the best I can do from any state WEIGHTED by the probability of reaching any state

One-state-MDP (known as k-Armed Bandit)

$r(a_i) = N(\mu_i, \sigma_i)$ then we can update the Q table with this rule:

$$Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha\left[r_{t+1}(a_i) - Q_{n-1}(a_i)\right]$$

error: actual reward and the predicted value

If we put $\alpha = 1/V_{M-1}(a_i)$ where $V_{M-1}(a_i)$ is the number of executions of action a_i up to time $M-1$, the above part of the updating rule is the on-line average of the reward that you get.

To choose an action we can again use the ε-greedy:

- uniform random choice with prob. ε (exploration)
- best choice with propability $1-\varepsilon$ (exploitation)

BALANCING EXPLORATION AND EXPLOITATION IS THE BEST TO IMPROVE PERFORMANCE ON AVG

What happens if parameters of Gaussian distribution slightly varies over time, $\mu_i \pm 10\%$ at UNKNOWN INSTANTS OF TIME?

[redacted]

THE RL IS STILL GOOD BUT YOU STILL HAVE THE CHOICE OF HOW DEFINE α. If the α is defined as before, the average of a many values from a distribution that changes is not good. A much better approach IS TO KEEP α CONSTANT (we have to assume to have a semi-stationary system → low frequent changes, much less frequent than trials);

With CONSTANT α THE SYS WILL ADAPT TO CHANGING CONDITIONS. Also with α constant my sys gives more importance to recent value of rewards

In the general case of multiple states the update rule is:

$$\hat{Q}_M(x,a) \leftarrow \hat{Q}_{M-1}(x,a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}_{M-1}(x',a') - \hat{Q}_{M-1}(x,a)\right]$$

This updating rule can be rewritten as:

$$\hat{Q}_M(x,a) \leftarrow (1-\alpha)\hat{Q}_{M-1}(x,a) + \alpha\left[r + \gamma \max_{a'} \hat{Q}_{M-1}(x',a')\right] \quad (3)$$

α is a parameter that balances how much you want to consider the error and the previous value in the Q-table

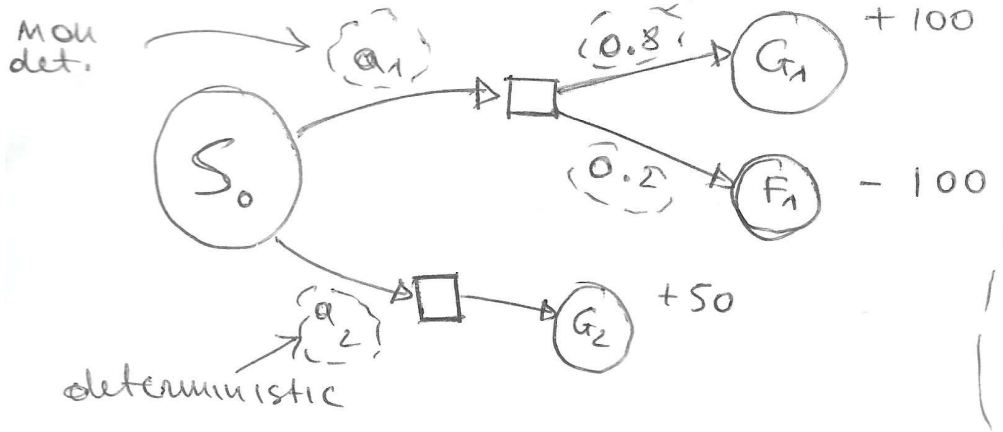
Here α can be constant or equal to $\frac{1}{1+V_{M-1}(x,a)}$ where

V_{M-1} = total number of times state-action pair (x,a) has been visited up to M-th iteration.

The Q-learning algorithm is the **same** of the deterministic case, the difference is only in the update rule.

- IN NON-DET. MDP IS NOT TRUE ANYMORE THAT THE Q-VALUES INCREASE OVER TIME (sometimes I can get a bad reward!)
- NON-DET Q-learning CONVERGES WHEN EVERY PAIR STATE-ACTION IS VISITED INFINITELY OFTEN (the algorithm can choose any action from any state).

Let's comment more in details what changes in the non deterministic case. Imagine to have an MDP with 4 states and 2 actions!



The agent can decide between a_1 and a_2, but HAS NO CHOICE ON THE OUTCOME OF a_1, it depends on the env. or by an opponent.

Suppose that you know the model, all the numbers in the above figure, which is the optimal policy? Let's compute the expected values:

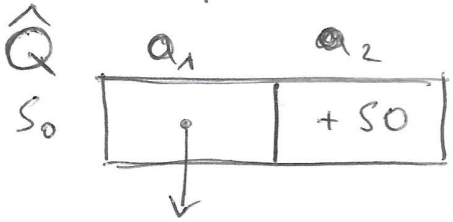
$$E[V^{a_1}(S_0)] = 0,8\,(+100) + 0,2\,(-100) = 60$$

$$E[V^{a_2}(S_0)] = 1\cdot(50) = 50$$

In this case the best policy is just the best action to

The optimal policy is a_1. In this case the MDP is known, but what happens if you don't know the "numbers" we cannot ask to which is the best policy. IN CASE YOU DON'T KNOW THE NUMBERS YOU HAVE TO SAMPLE THE MDP WITH TRIALS. (you have to do it many times).

We have to apply some reinforcement learning algorithm and you will update the Q-table, that is very simple!



WE HAVE TO MAKE TESTS. a_2 will always recieve an immidiate $R = 50$ and this will never change.

Here you have a set of values:

$R^{(1)}$ $R^{(2)}$ ... $R^{(M)}$

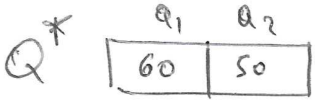
+100 −100 ... +100

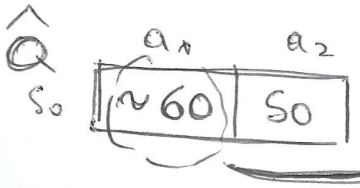
→ Rewards of different experiments and if we make enough of them, in the avg we converge to!

$\sim 80\%$ of $R^{(i)} = +100$

$\sim 20\%$ of $R^{(i)} = -100$

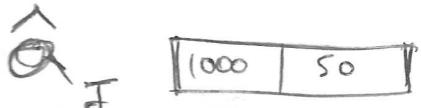
IF WE SAMPLE ENOUGH THE SYSTEM. When $M \to +\infty$ you will have these values and you will compute the expected value that is mean 60.





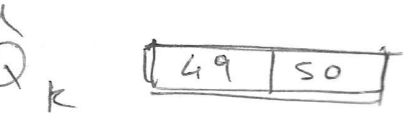
IF WE DO ENOUGH EXPERIMENTS.

We can accept some errors in the Q-function, provided THAT THE OPTIMAL POLICY IS THE SAME (both in $\hat{Q}$ and Q^* is a_1). In this case we do not necessary to reach 60, it is enough that is greater than 50. Suppose to have:



WRONG PREDICTION BUT WE GET OPT π!

VS

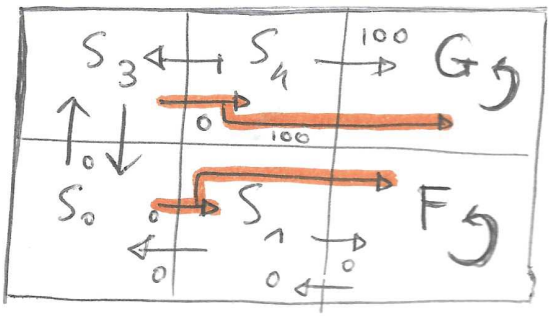


better algorithm, near to Q^* (49 near to 60) but DOES NOT GET THE OPT π!

THE REAL GOAL IS TO APPROXIMATE THE ARGMAX over all the possible actions for each state OF THE REAL Q-function

$$\underset{a}{\operatorname{argmax}}\ \hat{Q}(x,a) \overset{?}{=} \underset{a}{\operatorname{argmax}}\ Q^*(x,a)$$

Let's consider the following case:



Here there is only one opt π since it is more convenient to move up from S_0, so that I may reach the goal in two steps.

Let's consider other reinforcement learning methods:

TEMPORAL DIFFERENCES

The problem of the **$\hat{Q}$ update** is related to the term that tries to predict what is the best I can do in the future (is very noisy)

$$\hat{Q} \leftarrow r_t + \gamma \max_{a'} \hat{Q}$$

$\hat{Q}$ → is an approximation of Q

is good only if $\hat{Q}$ is a good approximation of Q

Temporal differences ⟶ bigger look ahead

I CAN COMPUTE n STEP TIME DIFFERENCE, I can collect the rewards of different steps and consider what I can do in the future:

$$Q^n(x_t, a_t) = r_t + \gamma r_{t+1} + \dots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(x_{t+n}, a)$$

The reward are discounted.

You can compute the 1-step difference, the 2-step difference, the M-step difference and do the merge.

What happens is that the agent does M steps, collects rewards in a memory, then it goes back in the past and updates the Q table for the original pair (x, a) where it started.

The merge is a weighted average by a parameter λ, this method is called $TD(\lambda)$:

$$Q^{(\lambda)}(x_t, a_t) = (1-\lambda)\left[Q^{(1)}(x_t, a_t) + \lambda Q^{(2)}(x_t, a_t) + \dots\right]$$

Q-learning is a special case of $TD(\lambda)$ in which $\lambda = 0$; the more $\lambda \geq 1$ the more you consider ~~[illegible]~~ backahead.

(+) Sometimes it converges faster than Q learning.

SARSA

What I can try to estimate is what I can do with my current policy

Key idea: INSTEAD OF COMPUTING THE BEST I CAN DO WITH π^* TRIES TO COMPUTE THE BEST I CAN DO WITH THE CURRENT POLICY.

The difference is that I do not use the max operator:

$$\hat{Q}_M(x,a) \leftarrow \hat{Q}_{M-1}(x,a) + \alpha\left[M + \gamma \hat{Q}_{M-1}(x',a') - \hat{Q}(x,a)\right]$$

This way of computing updates is also called on-policy method. THIS METHOD WILL NOT APPROXIMATE Q BETTER, but ALLOW TO COMPUTE THE POLICY BETTER.

· Explicit representation of $\hat{Q}$ table MAYBE NOT FEASIBLE FOR LARGE MODELS. Another aspect that is important:

> IN ORDER TO ASSIGN A VALUE, you have to visit a particular state-action-pair. To converge (in principle) you have to visit each state action pair at least once.

VERY DEMANDING REQUIREMENT, no generalization on unseen state action pairs.

We can use an approximation function of Q and learning this function is a regression problem.

$$Q_\theta(x,a) = \theta_0 + \theta_1 F_1(x,a) + \dots + \theta_M F_M(x,a)$$

Another possibility is to use NEURAL NETWORK and learn Q with the backpropagation.

-NOTE:
RL ⟷ (related) Supervised learning ⟼ in particular to Regression: LEARNING A REAL FUNCTION

Policy Iteration

Policy iteration is based on generating the policy in a parametric form and what you need are only the parameters. A typical approach is the policy gradient, compute the gradient and move in order to improve the value of this policy.

Parametric representation of π: $\pi_\theta(x) = \max_{a \in A} \hat{Q}_\theta(x,a)$

IDEA: Search in the POLICY-SPACE.
$p(\theta)$ = expected value of executing π_θ
policy gradient: $\Delta_\theta p(\theta)$

Choose θ ← initial parameter
WHILE termination condition
estimate $\Delta_\theta \rho(\theta)$
$\theta \leftarrow \theta + \eta \Delta_\theta \rho(\theta)$ ⟹ you generate a new policy that goes toward the positive gradient
END WHILE.

LOCAL MAXIMA
(greedy approach)

Since you don't know the function, you can only compute the gradient esperimentally. Suppose to have:

$$\pi_\theta = \vartheta_1(f_1(x)) + \vartheta_2(f_2(x))$$

All the functions that exhact information from the state

You parametrize each policy in this way

To evaluate the gradient you can add permutation of the parameters, adding some value ε, so you get another policy closer to the initial one and you can try it on the environment and test it. More general:

1. generating PERTURBATIONS of π_θ BY MODIFYING PARAMETERS
2. EVALUATE these PERTURBATIONS
3. GENERATE a NEW policy FROM "BEST SCORING" PERTURBATIONS.

(+) In this approach no representation of states, Works well if X is continuos (X is not always discrete)

(+) Reduce number of parameters.

$\pi \leftarrow$ initial policy

while termination condition

compute $\{R_1, \ldots, R_t\}$ random perturbations of π

evaluate $\{R_1, \ldots, R_t\}$

$\pi \leftarrow$ getBestCombination of $(\{R_1, \ldots, R_t\})$

end while

$$R_i = \{\theta_1 + \delta_1, \ldots, \theta_N + \delta_N\}$$

$$\delta_1 \in \{-\varepsilon_j, 0, \varepsilon_j\} \qquad \boxed{\varepsilon_j < \theta_j}$$

Combination of $\{R_1, \ldots, R_t\}$ is obtained by computing for each parameter j:

- $Avg_{-\varepsilon j}$: avg score of all R_i with a neg. perturb.
- $Avg_{+\varepsilon j}$: " " " " " " a pos. perturb.
- $Avg_{0 j}$: " " " " " " a zero perturb.

Vector $A = \{A_1, \ldots, A_N\}$

$$A_j = \begin{cases} 0 & \text{if } Avg_{0j} > Avg_{-\varepsilon_j} \wedge Avg_{0j} > Avg_{+\varepsilon j} \\ Avg_{\varepsilon j} - Avg_{-\varepsilon j} \end{cases}$$

finally:

$$\boxed{\pi \leftarrow \pi + \frac{A}{|A|}\eta}$$