Reinforcement learning

Dynamic systems
The state of a system
Markov Decision Processes (MDP)
- The Markov property
- Solving an MDP
- One state MDP
- The general case
- Evaluating RL agents
- Learning in the non-deterministic case
- K-armed bandit
- Learning in the general case

Hidden Markov Models (HMM)
- Markov chain
- Chain rule in HMM
- Learning in HMM

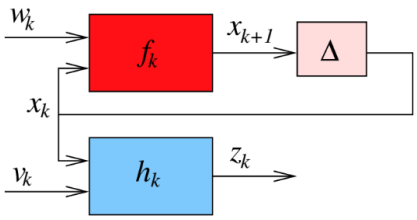
Partially Observable Markov Decision Processes (POMDP)

Dynamic systems

A dynamic system can be thought of as the composition of many interconnected units that evolves over time. This evolution is represented by the state of the system, that depends on the dymanics that make the system evolve.

When studying reinforcement learning, dynamic systems are characterized by the following parts:

- State X
- Observations Z
- Noise w,v
- State transition model f
- Observation model h



When studying a dynamic system we have two possible approaches, that depend on the informations that are available to us.

If we assume that all the informations about the system are exposed, then we work with **reasoning**. In fact having all the informations lets us know predict the future states of a system without making any action.

When we don't have a fully observable model of the system, what we can do is **learning** by executing actions in the environment and try to reconstruct the model.

The state of a system

The state of a dynamic system can be considered as a snapshot of the system at a given time (imagine that we can freeze the system for an instant and read its internal state).

A state is fully observable if at any given time an agent can fully read it. In particular, the task of an agent is to choose the best action to perform for each state. In particular, we want to define a **policy** function that maps states to actions:

$$\pi = X \to A$$

Where X is the set of all the possibe states and A the set of all the possible actions. When the model of the system is not known we will have to learn the policy.

The evolution of a system can be encoded as a set of sequences of states, actions and rewards that are updated during learning:

$$D = \{< x_1, a_1, r_1, x_2, ..., x_n, a_n, r_n >^{(i)}\}$$

Rewards are basically signals that the agent receives after having performed an action in a certain state. We use rewards to make the agent learn.

Markov Decision Processes (MDP)

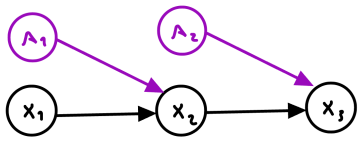
The Markov property

The Markov property is a fundamental property based on conditional independence between states. In particular it says that we just need to know the state and action at time t in order to compute the state at time t+1. In other words, the probability of the future depends only on the current state:

$$P(future|present, past) = P(future|present)$$

The models based on the assumptions made so far are called **Marvov Decision Processes (MDP)**.

We represent graphically these processes using the notation of Bayesian networks:



Here for example $P(x_2|x_1, a_1)$ and $P(x_3|x_2, a_2)$.

The formal definicion of an MDP is:

$$MDP =< Z, A, \delta, r >$$

Where:

- X is a finite set of states
- A is a finite set of actions
- $\delta : X \times A \rightarrow X$ is the transition function
- $r : X \times A \rightarrow \mathbb{R}$ is the reward function

The computations of states and rewards can be defined using the Markov assumption as:

$$x_{t+1} = \delta(x_t, a_t)$$
$$r_t = r(x_t, a_t)$$

In particular transitions can be of two types. **Deterministic transitions** map a state action pair to one possible evolution. Instead **non-deterministic transitions** happen when instead of having a fixed transition, we have a set of possible states that we might reach with a certain probability.

In the non-deterministic case, the transition function will be based on a cerain probability distribution in order to compute the next state. In other words we will have a stochastic representation of the transition function:

$$\delta = P(x_{t+1}|x_t, a_t)$$

In order to obtain an evolution $< x_1, a_1, r_1, x_2 >$, three steps are needed:

- The policy determines the action to execute based on the current state $a_1 = \pi(x_1)$
- The reward function computes the reward $r_1 = r(x_1, a_1)$
- The transition function determines the next state $x_2 = \delta(x_1, a_1)$

Solving an MDP

Solving a Markov Decision Process means finding the optimal policy, namely the best behaviour for the agent. In particular, the best behaviour is the one that maximizes the reward in the shortest time.

How do we define the reward?

In order to have the best reward in the shortest time we introduce the discount facotr $\gamma < 1$. Its role is to penalize the rewards over time, in fact we will consider $r_1 + \gamma r_2 + \gamma^2 r_3 + ...$

If the process that we are studying is non deterministic, we will consider the expected value of all. the rewards $E[r_1 + \gamma r_2 + \gamma^2 r_3 + ...]$.

This is also called **cumulative discounted reward**.

We can use the cumulative discounted reward as a metric for measuring how good is the policy. We define the value function of a policy as:

$$V^{\pi} = E[r_1 + \gamma r_2 + \gamma^2 r_3 + ...]$$

In other words, our goal will be to find the policy that maximizes the value function. We say that π_1 is better than π_2 if:

$$V^{\pi_1}(x_1) > V^{\pi_2}(x_1)$$

The optimal policy is the one with the highest value function (there could be more equivalent optimal policies):

$$V^{\pi^*}(x_1) \geq V^{\pi}(x_1) \;\; \forall x_1 \in X$$

We denote the value of the optimal policy as $V^*(x_1)$.

One state MDP

We start by considered an MDP where we have one state x_0 and many possible actions that always lead to x_0:

This is also defined as the bandit problem, since it remembers the dynamics of a slot machine with n available levers to pull (actions). Each lever will produce a certain reward.

Let's now consider some different situations we could find ourselves in.

The rewards are deterministic and known

In this case we have a table that associates each action to a reward. Here the optimal policy is simply to choose the action that will return the maximum rewards, hence zero trials are needed.

$$\pi(x_0) = argmax_{a_i}\{r_i\}$$

The rewards are deterministic, but not known

Here we just know the available actions. The optimal strategy is to try each action and observe the reward. We are guaranteed to find the optimal policy by performing n trials in total since the reward is still deterministic. Therefore we will have the same formulation for the optimal policy:

$$\pi(x_0) = argmax_{a_i}\{r_i\}$$

The reward is known, but non-deterministic

We can think this situation as having a table that associates each action to a certain probability distribution for the reward in terms of mean and variance. The best strategy will be to choose the distribution with the highest mean:

$$\pi(x_0) = argmax_{a_i}\{\mu_i\}$$

This strategy takes 0 trials.

The reward is non-deterministic and not known

Here the rewards is stochastic and its distribution is not known. A first strategy we can imagine is to repeat each action for a certain amout of trials and compute the means for each action. But this is not really an optimal strategy.

A better approach is to compute the means *online*. This means that we will update the value of the means in real time and update the strategy accordingly. In particular we will perform an action at each step and update the mean by collecting the reward. In practice we will implement an algortithm that will update a data structure at each step like this:

1. Initialize the data structure Θ
2. For each time $t = 1, ..., T$ (until termination condition):
    a. Choose an action $a_t \in A$
    b. Execute a_t and collect the reward r_t
    c. Update the data structure Θ
3. Return the optimal policy $\pi^*(x_0)$ according to Θ

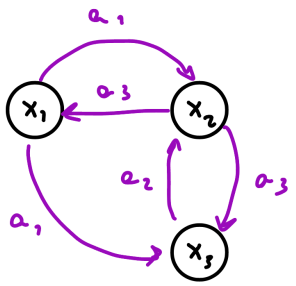
If we assume that the reward follows a Gaussian distribution, we can define the algorithm as follows:

1. Initialize $\Theta_{(0)}[i] \leftarrow 0$ and $c[i] \leftarrow 0$ with $i = 1, ..., |A|$
2. For each time $t = 1, ..., T$:
    a. Choose an index $\hat{i}$ for action $a_t = a_{\hat{i}} \in A$
    b. Execute a_t and collect the reward r_t
    c. Increment $c[\hat{i}]$
    d. Update $\Theta_t[i] \leftarrow \frac{1}{c[\hat{i}]}(r_t + (c[\hat{i}] - 1)\Theta_{t-1}[\hat{i}])$
3. The optimal policy will be $\pi^*(x_0) = a_i$ with $i = argmax_{i=1,...,|A|}\Theta$

This is the skeleton of the algorithm that we will actually implement. An important feature of the online approach is that it can adapt even to situations where the reward changes over time.

The general case

In the general case we consider multiple states, for example:



When we don't know both the transition and reward functions, what we can do is learning through executing actions and collecting data about the reward obtained and the next state.

A first approach to learning is *value iteration*. We start from the notion that the best action to execute in a certain policy depends on the optimal policy:

$$\pi^* = argmax_{a \in A}[r(x, a) + \gamma V^*(\delta(x.a))]$$

Notice that the current formulation depends on functions we don't have. But *do we really need them?*

What we really want is the quantity expressed by these function. We define it using the Q function:

$$Q^*(x, a) = r(x, a) + \gamma V^*(\delta(x.a))$$

If we find the optimal Q function, then the problem is basically solved. What we want is:

$$\pi^*(x) = argmax_a\{Q^*(x,a)\}$$

We will start with an underestimate for the optimal Q function and try get as close as possible to it over time.

Even if at first we don't know it, we have to remember that each state has its own value. The Q function will express the value of that state as the expected reward if we used the optimal policy.

When we start from scratch, the strategy is to assign a value of 0 to every state. As the iterations go on, we will continuosly update the values after executing some action. A fundamental assumption of this strategy is that the rewards are all non-negatives ($r(x,a) \geq 0$). If the system has negative rewards, we simply re-scale them. Each time that an action is performed in a certain state, the corresponding reward will become explicit, and we will exploit this knowledge to learn.

Learning the Q function: the deterministic case

Since we don't know the optimal Q function a priori, we start from an underestimate $\hat{Q}$, that we will evaluate this way:

1. Fix a state x_1 and perform all the possible actions
2. Collect the rewards associated to each state
3. The best action will be the one that produced the highest reward
4. Update $\hat{Q}(x_1,a) \leftarrow r(x_1,a) + \gamma max_{a' \in A}\hat{Q}(x',a')$

At each step the value on the underestimate will keep increasing and tend to the optimal value Q^*.

The notation that we use for the Q function associated to a certain policy π is $Q^{\pi}(x,a)$, while for the optimal function, we only use $Q(x,a)$.

Notice that:

$$V^*(x) = max_{a \in A}\{r(x,a) + \gamma V^*(\delta(x,a))\} = max_{a \in A}Q(x,a)$$

Hence, holds the equality $Q(x,a) = r(x,a) + \gamma V^*(\delta(x,a))$ that lead us to the following formula:

$$Q(x,a) = r(x,a) + \gamma\ max_{a \in A}Q(\delta(x,a),a')$$

Meaning that the optimal Q function depends from the immefiate rewards and the value that maximized the Q function. This helps us to define the training rule as:

$$\hat{Q}(x,a) \leftarrow \bar{r} + \gamma\ max_{a'}\hat{Q}(x',a')$$

The skeleton of the learning algorithms is:

1. For each possible state-action pair (x,a) initialize a table entry $\hat{Q}_{(0)}(x,a) \leftarrow 0$
2. Observe the current state x
3. For each time $t = 1, ..., T$:
    a. Choose an action a
    b. Execute a
    c. Observe the new state x'
    d. Update the entry of the Q underestimate as $Q_{(t)}(x,a) \leftarrow \bar{r} + \gamma\ max_{a' \in A}\hat{Q}_{(t-1)}(x',a')$
    e. $x \leftarrow x'$
4. Return the optimal policy $\pi^*(x) = argmax_{a \in A}\hat{Q}_{(T)}(x,a)$

Since the reward is zero, we will be guaranteed that our estimate will improve over time:

$$0 \leq \hat{Q}_{(n)}(x,a) \leq \hat{Q}_{(n+1)}(x,a) \leq Q(x,a)$$

The convergence will be guaranteed if each state-action paire is visited infinitely often, meaning that the possibility of performing every available action is non zero.

How to choose which action to perform?

Here we have two possible stratiegies:

- **Exploitation**: select each time the action that maximizes the value $\hat{Q}(x,a)$ (if known)

- **Exploration**: select the action randomly

In pratice, we have to find a tradeoff between exploration and exploitation. In facti while exploitation guarantees to seek the maximum value, exploration lets us explore more possible (better) paths.

How to find the right balance?

A possible solution is to implement an ϵ-greedy strategy. Given a parameter $0 \leq \epsilon \leq 1$, we select a random action with probability ϵ, or the best action with probability $1 - \epsilon$.

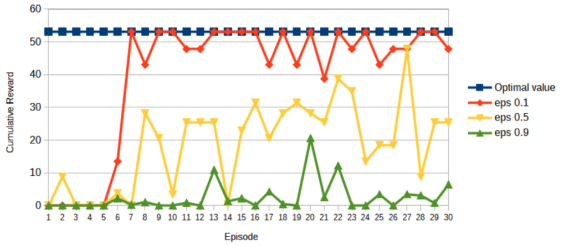
A common aproach is to start with a high value of ϵ in order to favour exploration, and decrease it over time.

Another strategy is softmax, that favours the choice of the probability of choosing the avtion with the higher value.

Evaluating RL agents

Let's suppose that we trained an agent. How do we evaluate the policy he learned?

A possibility is to plot the cumulative reward as the agent learns. Since we have an alternation between exploration and exploitation, the resulting that can be very noisy:



A more useful approach consists in executing k steps of learning an then evaluating the current policy based on the average of the cumulative rewards obtained.

Moreover, other domain specific metrics can be defined.

Learning in the non-deterministic case

In the non-deterministic case actions can lead to different possible states with a certain probability.

In this case it makes sense to consider the expected value of the cumulative discounted reward:

$$V^{\pi}(x) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + ...] = E[\sum_{i=0}^{\infty} \gamma^i r_{t+1}]$$

The optimal policy will be the one that maximizes the expected value:

$$\pi^* = argmax_{\pi} V^{\pi}(x) \; \forall x$$

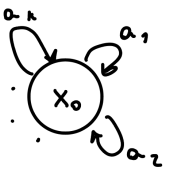
From this observations, we can derive the following equality:

$$Q(x,a) = E[r(x,a) + \gamma V^*(\delta(x,a))] = ... = E[r(x,a)] + \gamma \sum_{x'} P(x'|x,a) V^*(x') = E[r(x,a)] + \gamma \sum_{x'} P(x'|x,a) m$$

The optimal policy will be $\pi^*(x) = argmax_{a \in A} Q(x,a)$.

K-armed bandit

Imagine that we have one state x_0 and k possible levers to pull (actions) $a_1, ..., a_k$:



Let's assume that the reward is stochastic and follows a Gaussian distribution:

$$r(a_i) = \mathcal{N}(\mu_i, \sigma_i)$$

In general we know that it's best to choose the action that corresponds to the reward with the maximum mean. But what if we don't know the parameters of the distributions?

We can approximate the distribution by computing an average reward and update it at each new step. The training rule is:

$$Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha[\bar{r} - Q_{n-1}(a_i)]$$

Where $\alpha = \frac{1}{1+v_{n-1}(a_i)}$ and v_{n-1} the number of execution of action a_i up to time n-1.

Learning in the general case

In the general case we have many actions associated to many possible states. Here we have to adapt the previous logic to the need of taking into account many states:

$$\hat{Q}_n(x,a) \leftarrow \hat{Q}_{n-1}(x,a) + \alpha[r + \gamma\, max_{a'}\hat{Q}_{n-1}(x',a') - \hat{Q}_{n-1}(x,a)]$$

Where the term $r + \gamma\, max_{a'}\hat{Q}_{n-1}(x',a')$ represents the cumulative average over the state.

Whle the approaches we have seen so far are valid, the actual functions used in learning are the parametrized versions of the previous ones. In fact the Q value can be represented as a functions instead of tabular data:

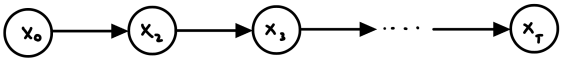
$$Q_\theta(x,a) = \theta_0 + \theta_1 F_1(x,a) + ... + \theta_n F_n(x,a)$$

An important feature of this representation is that it allows us to take in account similarities between states. For example, is a state gives an high reward, it is more likely that states near it can be considered good as well. Morevore, assigning weights to the Q functions allows us to learn by exploiting deep learning (Deep Reinforcement Learning), that allows us to deal with the continuous case as well.

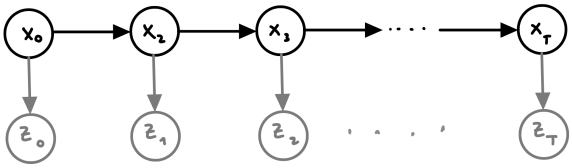
Hidden Markov Models (HMM)

Markov chain

The evolution of a dynamic system can be modeled as a random process based on the markov assumptions. The graphical representation of these states is called markov chain:



In many cases we have to deal with situation where the actuale value of the state is not known (we say that the state is not observable). What the agent can do is observe it from the outside through some obervations, that can be modeled as random variables $z_0, z_1, ..., z_T$:

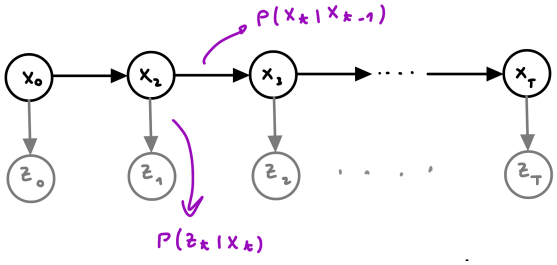


Problems that are modeled in this ways are called Hidden Markov Models. HMMs are formally defined as a set of states X, observations Z and initial distribution π_0:

$$HMM =< X, Z, \pi_0 >$$

Here we can't influence the evolution of the system because we don't have access to actions. What we have is.

- A transition model $P(x_t|x_{t-1})$
- An observation model $P(z_t|x_t)$
- An initial distribution over the states π_0



The probability distribution of being in a certain state given that we were in another state at the previous step can be modeled as a matrix. For example, let's suppose that we have three state $X = \{a, b, c\}$. Here the transition model is represented as $P(x_t = \{a, b, c\} | x_{t-1} = \{a, b, c\})$.

The matrix model is:

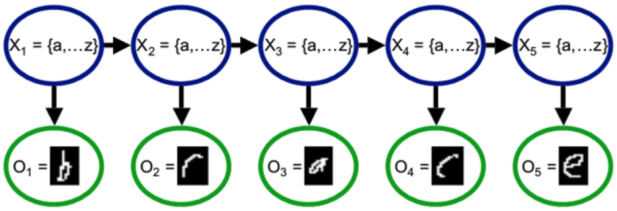
	$x_{t-1}=a$	$x_{t-1}=b$	$x_{t-1}=c$
$x_t=a$	:	0.3	0.1
$x_t=b$	:	0.5	0.2
$x_t=c$	:	0.2	0.1

Let's assume that we have two observations available $Z = \{\alpha, \beta\}$, the relation between them and the actual states can be encoded in an observation matrix:

	$x_t=a$	$x_t=b$	$x_t=c$
$z_t=\alpha$	0.1	0.3	..
$z_t=\beta$	0.4	0.7	..

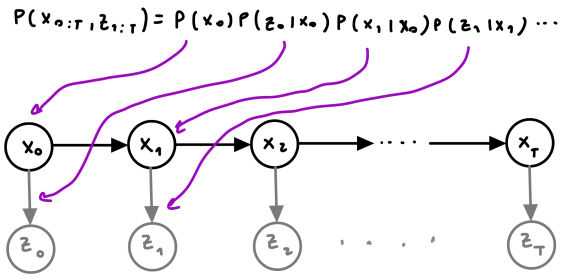
In cases where the state is modelled as a Gaussian distribution, meaning that $x_t = \mathcal{N}(\mu_t, \sigma_t)$, then observations are continuous and modelled as $P(z_t | x_t = \{a, b, c\})$ (the observations will follow a certain gaussian distribution, that will depend on the state).

The power of hidden markov models is that they can model a problem by analyzing the evolution of the system. For example in the following image we see that the caracter $O_2 = r$ and $O_4 = c$ are written in a similar way. But using HMM, we can take into account that given the previous letter, it is more likely to have a c in place 4 (brace is a word that makes sense, brare not).



Chain rule in HMM

Since HMM ar ebased on the Markov's assumption of conditional independence, computing the probability of being at state T given a certain observation can be computing by applying the chain rule as illustrated here:



In particular, when the model of an HMM is known, we can compute all these informations directly.

Two important problems where HMMs are applied are:

- **Filtering**: estimate the state at time T given the previous states $P(x_T = k|z_{1:T})$
- **Smoothing**: given all the observation done until time T, find the value of a certain state in the past.

Learning in HMM

In many cases it happens that we don't know the model of the problem. A first possible case is the following: let's assume that we have an initial phase of the training where we can observe the actual state. Here we can simply estimate the value of all possible cases as:

Transition model computed as i,j transitions vs all the other possible transitions

$$A_{ij} = \frac{|i \to j \text{ transitions}|}{|i \to k \text{ transitions}|}$$

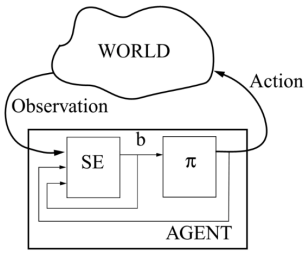
Observation model computed by considering all the time we made an observation v and the state is k, vs all the other observations in that state.

$$b_k(v) = \frac{|\text{observe } v \wedge \text{state } k|}{|\text{observe } k \wedge \text{state } k|}$$

If we can't observe the state during training, then what we can do is learning through observation by using the expectation maximization strategy.

Partially Observable Markov Decision Processes (POMDP)

In the POMDP case we have a dynamic system whose underlying evolution is modeled my a markov model. The state is only partially observable, and we can execute actions in order to drive the evolution of the system. Basically, this problem can be considered as a union between MDP and HMM.



A formal definition of this model is that $\text{POMDP} =< X, A, Z, \delta, r, O >$, where:

- X is the set of all possible states
- A is the set of all possible actions
- Z is the set of all possible observations
- $P(x_0)$ is the probability distribution over the initial state
- $\delta(x, a, x') = P(x'|x, a)$ is the probability distribution of the transition between states
- $r(x, a)$ is the reward function

- $O(x', a, z') = P(z', |x', a)$ is the probability distribution over the possible observations

What is the solution of this model?

Here we want to fung the best policy, namely the function that maps states to actions and maximizes the reward. Our problem is that in POMDP we don't know the state.

But the thing we know is the hystory of actions and observations, thus it makes sense to express the policy as $\pi : \{z_0, z_1, ..., z_{t-1}\} \rightarrow a$. Notice that this function is difficoult to represent, since it would require to store a very high number of observation.

A better approach is to introduce the concept of belief (estimation) of the state. This means that we make an estimate of the current state based on the previous observations, and update it at time t in order to choose the next action. We obtain the function $\pi : b_t \rightarrow a_t$.

Once we defined the belief function, then a POMDP basically becomed an MDP.