An Introduction to Deep Reinforcement Learning

Part 1 – MDPs, Dynamic Programming, Q-Learning, Deep Q-Learning

Karim Bouyarmane

The *subject* of Reinforcement Learning are **Markov Decision Processes** (MDP)

More precisely, Reinforcement Learning is a **Machine Learning** approach **to solving MDPs**

MDP: simplest possible probabilistic model of “something” that can “take actions”/decisions and act on itself or on the world

$$\begin{tikzcd} s \arrow[rr, bend left, "a"] & & s' \arrow[d] \\ & & r \end{tikzcd}$$

agent/world with **states** $s \in \mathcal{S}$ and possible **actions** $a \in \mathcal{A}$

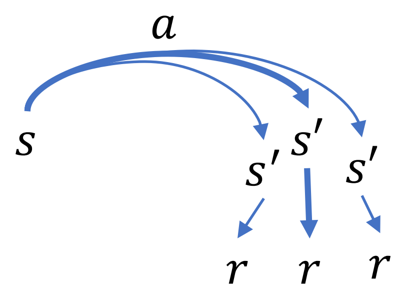
(e.g. physical robot, trading agent, video-game playing agent, continuous decision maker in dynamic and uncertain environment, etc.)

Two models and **one parameter** are necessary to fully characterize the MDP:

- a **transition model** $P(s'|s,a) = T(s,a,s')$ (a.k.a *dynamics model, “what is the effect of an action?”*)
- a **reward model** at state s : $R(s) \in \mathbb{R}$ (*“what is our objective? what state are we trying to reach?”*) (or $R(s,a)$, or even $R(s,a,s')$, etc.)
- a **discount factor** $0 < \gamma \leq 1$ (trade-off between immediate reward and delayed reward, “cost of delayed reward”)

MDP: simplest possible probabilistic model of "something" that can "take actions"/decisions and act on itself or on the world

agent/world with **states** $s \in \mathcal{S}$ and possible **actions** $a \in \mathcal{A}$

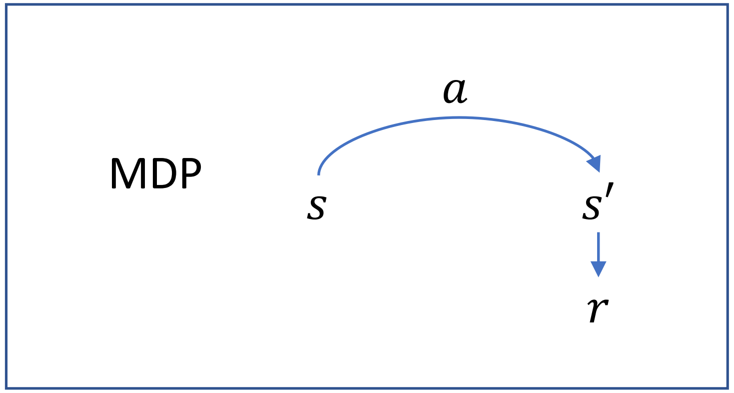


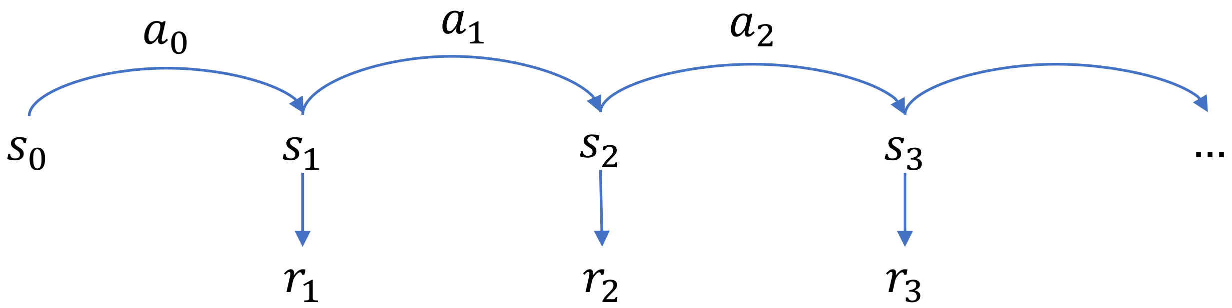
(e.g. physical robot, trading agent, video-game playing agent, continuous decision maker in dynamic and uncertain environment, etc.)

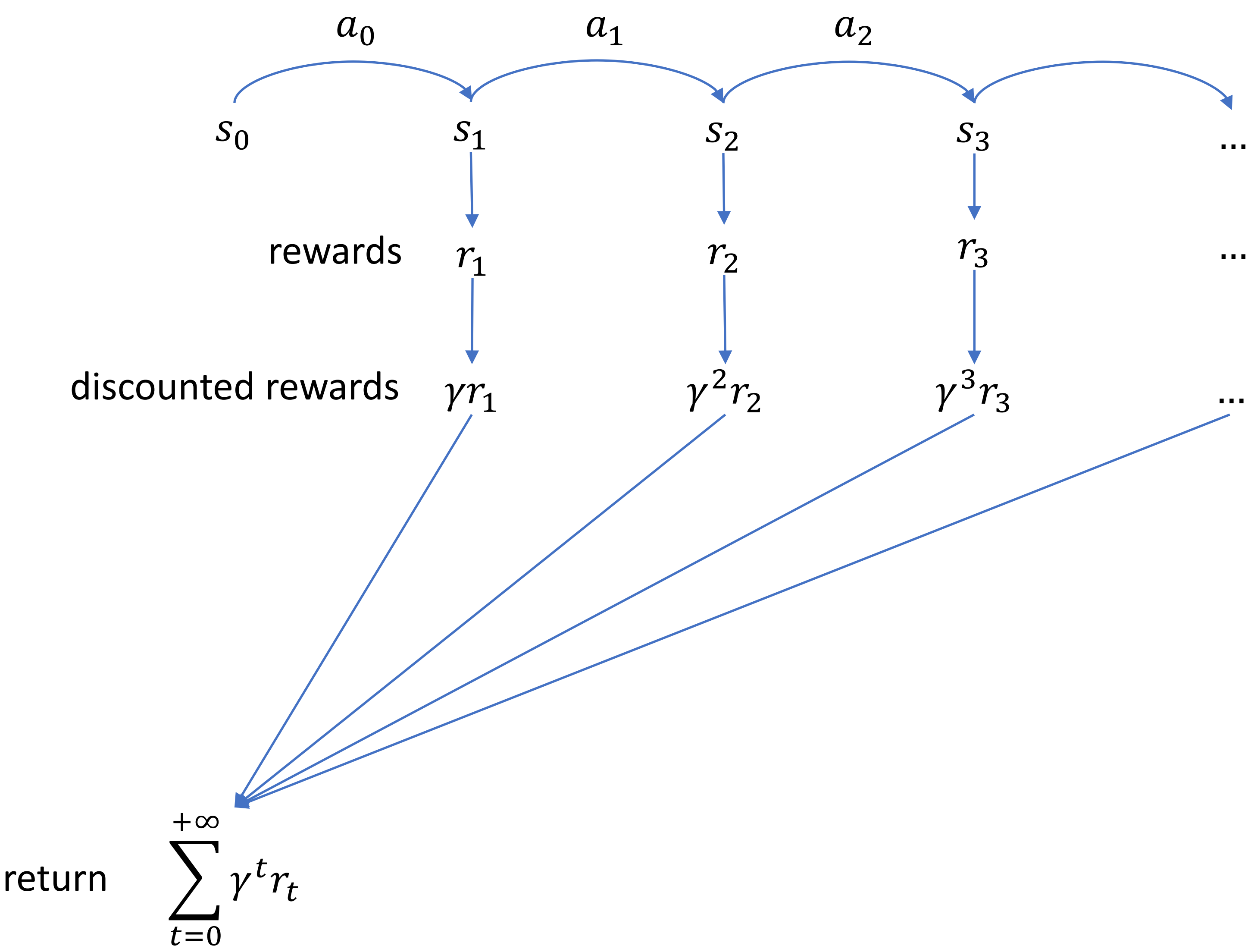
Two models and **one parameter** are necessary to fully characterize the MDP:

- a **transition model** $P(s'|s,a) = T(s,a,s')$ (a.k.a *dynamics model, "what is the effect of an action?"*)
- a **reward model** at state s : $R(s) \in \mathbb{R}$ (*"what is our objective? what state are we trying to reach?"*) (or $R(s,a)$, or even $R(s,a,s')$, etc.)
- a **discount factor** $0 < \gamma \leq 1$ (trade-off between immediate reward and delayed reward, "cost of delayed reward")

MDP = $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$


MDP
a
s
s'
r


a_0
a_1
a_2
s_0
s_1
s_2
s_3
...
r_1
r_2
r_3


a_0
a_1
a_2
s_0
s_1
s_2
s_3
...
rewards
r_1
r_2
r_3
...
discounted rewards
γr_1
$\gamma^2 r_2$
$\gamma^3 r_3$
...
return
$\sum_{t=0}^{+\infty} \gamma^t r_t$

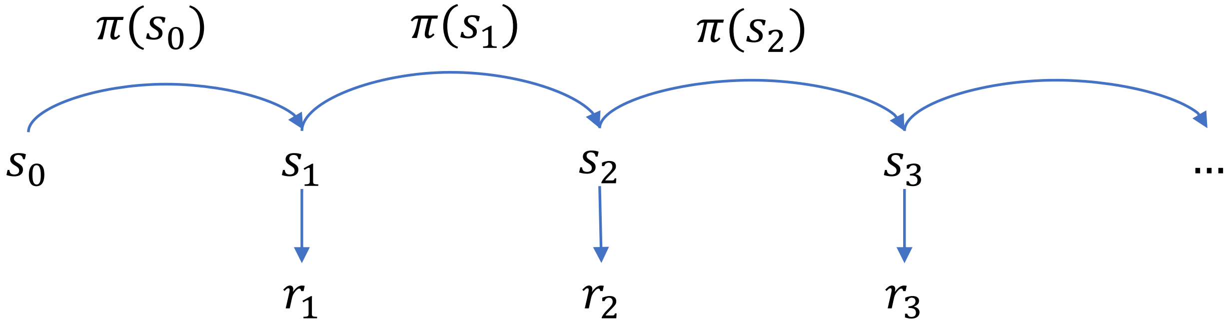
Policy = deciding what action to take at every state $\pi: s \mapsto a$

(a.k.a. "feedback loop", "control law", "control policy", "decision function", etc.)

"autonomous agent" = agent that follows ("is endowed with") a policy π

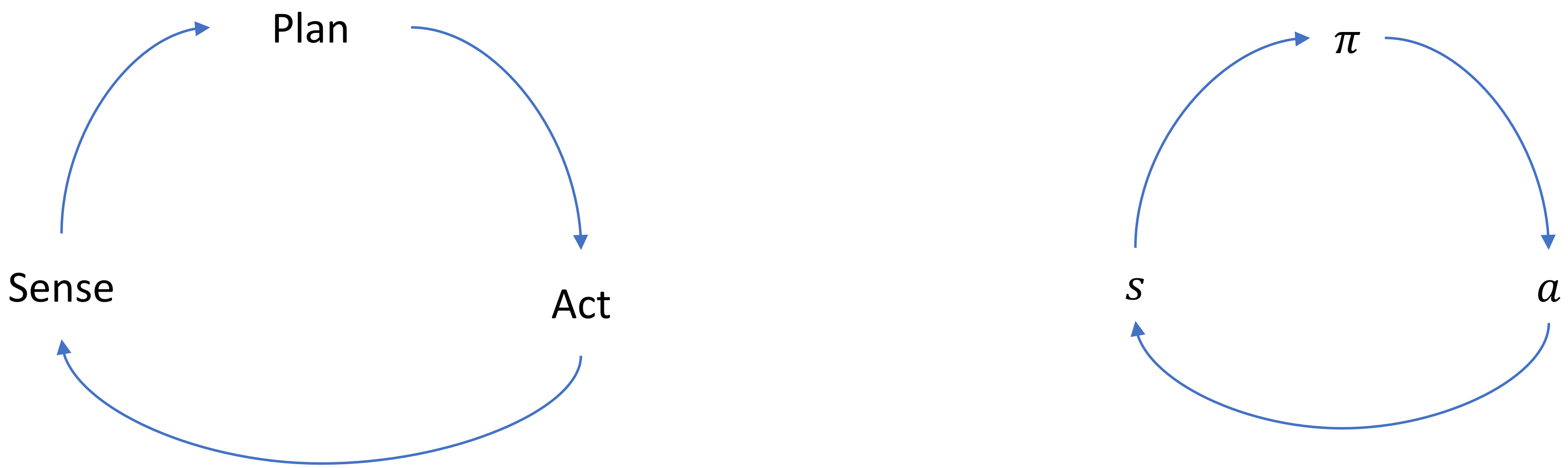
"Solving" an MDP = solving for a **policy** $\pi: \mathcal{S} \to \mathcal{A}$

$$\pi(s) = a$$



autonomous agent that follows a policy π

The autonomy feedback loop revisited



We don’t want to find *any* policy, we want to find a *good* policy

A *good* policy is a policy that takes actions that make the agent **maximize its long-term rewards** (or ***returns***), i.e. that makes the agent realize a certain *objective* (the objective being *encoded* in the reward/returns model)

The art of **formulating a good MDP** is thus **formulating a good reward model that captures the desired objective**

A good policy can also be interpreted a policy that **minimizes cost** ($cost = -reward$)

Examples of long term rewards:

- Winning a game
- Accomplishing a task successfully
- Reaching a goal position
- Making stock gains at a certain maximum horizon

Definition: A sequence of states s_t **follows a policy** π if

$$\forall t \geq 0, \qquad s_{t+1} \sim P(s_{t+1} | s_t, \pi(s_t))$$

We write $s_t \sim \pi$

So, we want to find the **optimal** policy π^*

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{s_t \sim \pi}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t) \ \mid\ \pi\right]$$

Where
$s_0 \sim$ given distribution
$s_{t+1} \sim P(s_{t+1} | s_t, \pi(s_t))$

Solving for the optimal policy is thus an **optimization problem (optimal control)** over the space of policies ($\mathcal{A}^{\mathcal{S}}$)

Different families of methods for solving MDP

- **Non-ML MDP Solving: Dynamic programming** methods
  - Value iteration
  - Policy iteration

- **Q-learning** methods **(DQN)**

- **Policy gradient** methods **(Actor-Critic)**

- **Evolution strategies** or DFO: Derivative-Free Optimization (**CMA-ES**)

Value of a state (or **Utility of a state**) V-value (or U-value)

$$V(s)\ (or\ U(s)) = \mathbb{E}_{s_t \sim \pi^*}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t)\ |s_0 = s, \pi^*\right]$$

Value of a state (or **Utility of a state**) V-value (or U-value):

“Best returns we can hope for in average, if we start from the state”

(meaning that we start from the state, and follow the optimal policy)

If we knew the V-value of every state, then the optimal policy at any given state is to take the action that gives you the best chance to land on the highest-value state

i.e. **optimal policy = "follow the V-values"**

If we knew the V-value of every state, then the optimal policy at any given state is to take the action that gives you the best chance to land on the highest-value state

i.e. **optimal policy = "follow the V-values"**

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') V(s')$$

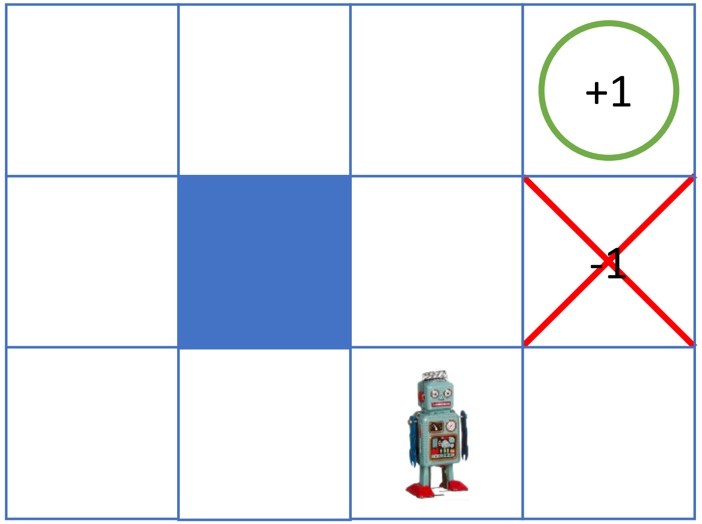
If we knew the V-value of every state, then the optimal policy is

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s\prime} T(s, a, s')V(s')$$

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t) \,|s_0 = s, \pi^*\right] \qquad \Leftrightarrow \qquad \pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s, a, s') V(s')$$

$$\text{“ } V = f(\pi^*) \text{ ”} \qquad \Leftrightarrow \qquad \text{“ } \pi^* = f^{-1}(V) \text{ ”}$$

Introducing Gridworld®



$\mathcal{A} = \{\uparrow, \leftarrow, \downarrow, \rightarrow\}$

$\mathcal{S} = \{1,2,3,4,5,6,7,8,9,10,11\}$

for every state $s \in \mathcal{S}$, let us denote

$s^{\uparrow}$ the state immediately to the north of s (if it exists)

$s^{\downarrow}$ the state immediately to the south of s (if it exists)

$s^{\leftarrow}$ the state immediately to the west of s (if it exists)

$s^{\rightarrow}$ the state immediately to the east of s (if it exists)

the transition model is

$$P(s^{\uparrow} | s, \uparrow) = 0.8 \qquad P(s^{\rightarrow} | s, \rightarrow) = 0.8$$
$$P(s^{\leftarrow} | s, \uparrow) = 0.1 \qquad P(s^{\uparrow} | s, \rightarrow) = 0.1 \qquad \ldots$$
$$P(s^{\rightarrow} | s, \uparrow) = 0.1 \qquad P(s^{\downarrow} | s, \rightarrow) = 0.1$$

+ If the robot bumps into a wall, it stays in the same state

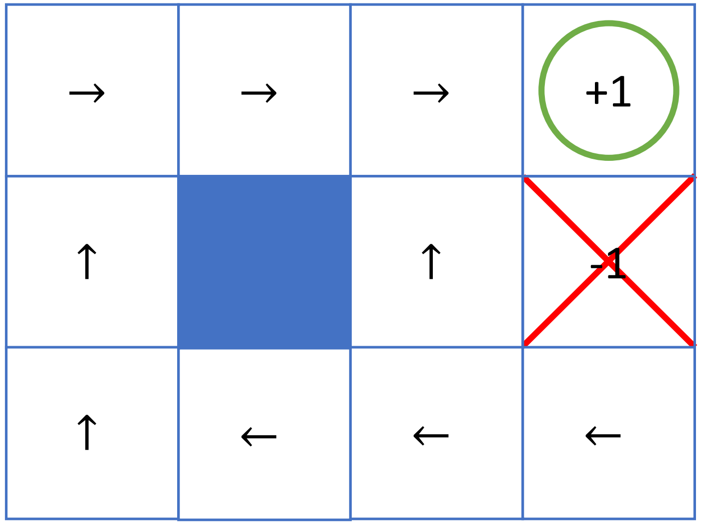
the reward model is

$$R(11) = +1$$
$$R(10) = -1$$
$$R(s \neq 10 \text{ and } 11) = -0.04$$

the discount factor is

$$\gamma = 1$$

Optimal policy π^*



Value of every state V

0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t) \mid s_0 = s, \pi^*\right] \quad \Leftrightarrow \quad \pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')V(s')$$

Value of every state V

0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

Optimal policy π^*

→	→	→	+1
↑		↑	-1
↑	←	←	←

The value function <u>or</u> **the optimal policy**, completely characterize the optimal solution of an MDP

Bellman equation:

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s') V(s')$$

Bellman equation:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')V(s')$$

In the example state (the one with value 0.918), if the agent follows the optimal action (which is “go to right”), then it has 80% chance of actually going to the right, landing in a state of value 1, 10% chance of going up, bumping into the wall, and thus stating in the same state with value 0.918, and 10% chance of going down, landing in the state of value 0.660, i.e.

$$0.8 * \boxed{1} + 0.1 * \boxed{0.660} + 0.1 * \boxed{0.918} - 0.04 = \boxed{0.918}$$

0.812	0.868	0.918	+1
0.762		0.660	~~-1~~
0.705	0.655	0.611	0.388

Bellman equation:

$$V(s_1) = R(s_1) + \gamma \max_a T(s_1, a, s_1)V(s_1) + T(s_1, a, s_2)V(s_2) + T(s_1, a, s_3)V(s_3) + \cdots$$
$$V(s_2) = R(s_2) + \gamma \max_a T(s_2, a, s_1)V(s_1) + T(s_2, a, s_2)V(s_2) + T(s_2, a, s_3)V(s_3) + \cdots$$
$$V(s_3) = R(s_3) + \gamma \max_a T(s_3, a, s_1)V(s_1) + T(s_3, a, s_2)V(s_2) + T(s_3, a, s_3)V(s_3) + \cdots$$
$$\vdots$$

Solving Bellman equation $\Rrightarrow$ Solving for V $\Rrightarrow$ Obtaining π^*

The Bellman equation

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s') V(s')$$

is a **fixed-point equation**

$$\text{“} \; V = Bellman(V) \; \text{”}$$

To solve a fixed-point equation, we apply the iteration method:

Initialize a random V_0

$$V_{k+1} = Bellman(V_k)$$

$$\lim_{k \to +\infty} V_k = V$$

Value-iteration (for finding the optimal policy of an MDP)

- Initialize $V_0(s)$ at some random values at all states s
    - Apply Iterative Bellman equation

$$V_{k+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s') V_k(s')$$

    - Loop
- Until convergence of $V_k(s)$ to some value $V(s)$
- Apply

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s') V(s')$$

There is another method very similar to Value-iteration, that solves directly for π^* called **Policy-iteration**

Reminder - Bellman equation:

$$V(s) = R(s) + \gamma \max_{a} \sum_{s'} T(s, a, s')V(s')$$

Linear Bellman equation, by definition of π^*:

$$V(s) = R(s) + \gamma \sum_{s'} T(s, \pi^*(s), s')V(s')$$

Policy-iteration (for finding the optimal policy of an MDP)

- Initialize $\pi_0(s)$ at some random values at all states s
    - Solve linear Bellman equation for V_k, given optimal policy π_k

$$V_k(s) = R(s) + \gamma \sum_{s'} T(s, \pi_k(s), s') V_k(s')$$

    - Update

$$\pi_{k+1}(s) = \operatorname*{argmax}_a \sum_{s'} T(s, a, s') V_k(s')$$

    - Loop
- Until convergence of $\pi_k(s)$ to some value $\pi^*(s)$

Value-iteration and Policy-iteration are **exact methods** to solve MDPs, they are not Machine Learning approaches

Why would we need Machine Learning to solve MDPs anyways?

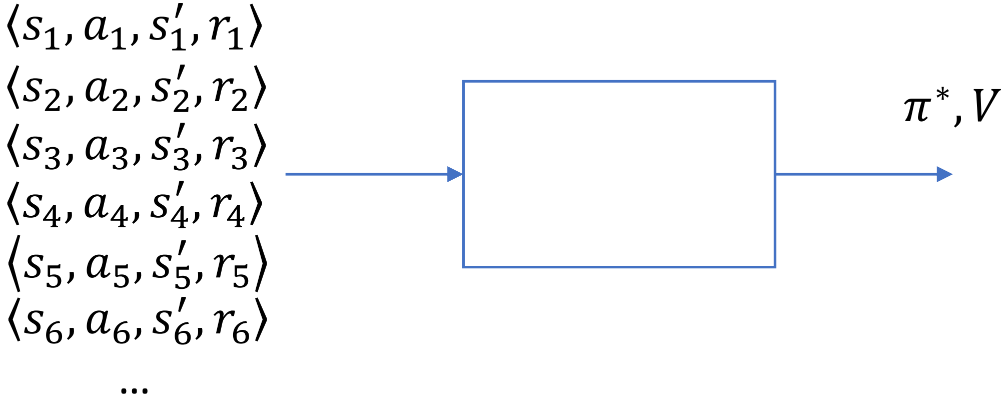
Usually we don't know the transition and reward model a priori, **we don't know $\boldsymbol{T}$ and $\boldsymbol{R}$**

We can only **observe** some **sample data** from T and R by making the agent actually perform in real world or simulate different actions a at different states s, then record what state s' we ended up in and what reward r we got as a result from that action at that state

Records of observed data from experiences will be in the form of **tuples $\langle \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', \boldsymbol{r} \rangle$**

Classical MDP solving: Model-based
input: model (T, R), output: policy

Reinforcement Learning for solving MDPs: Data-based
Input experience records (data) $\langle s, a, s', r \rangle$, output policy



Reinforcement learning template:

- Start with a random policy
    - Following this policy (exploitation) interleaved with some random actions from time to time (exploration), make the agent collect experience record tuples
    - Refine the policy based the knowledge received from these actions and these observations
    - Loop
- Until the policy converges

What is it exactly that we “learn”?

- **Not directly the model $\boldsymbol{T}$ and $\boldsymbol{R}$**, since we only care about the policy π
- **Maybe learn V-value of every state?** too coarse, we don’t have experience data directly associated with states s, but with actions a taken at state s
- We introduce a new quantity that refines V-values ⇛ by giving value to a pair of <action, state> the **Q-value of an action $\boldsymbol{a}$ at state $\boldsymbol{s}$**

V-value of a state:

$$V(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s')V(s')$$

Q-value of an action at a state:

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s')V(s')$$

Q-value of an action at a state:

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s')V(s')$$

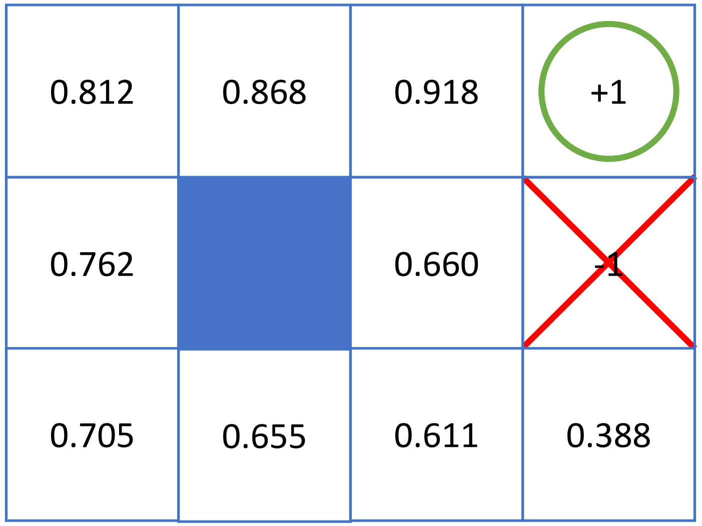
Q-value is also known as *action-value,* as opposed to *V-value* which is known as *state-value*

Q-value of an action at a state:

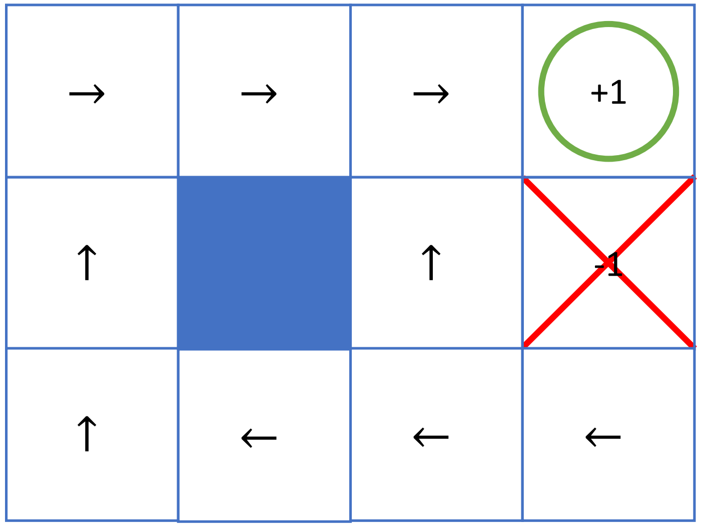
Best returns we can hope for if we take action a at state s

(meaning that we take action a at state s and then start following the optimal policy from whatever state s' we land at)

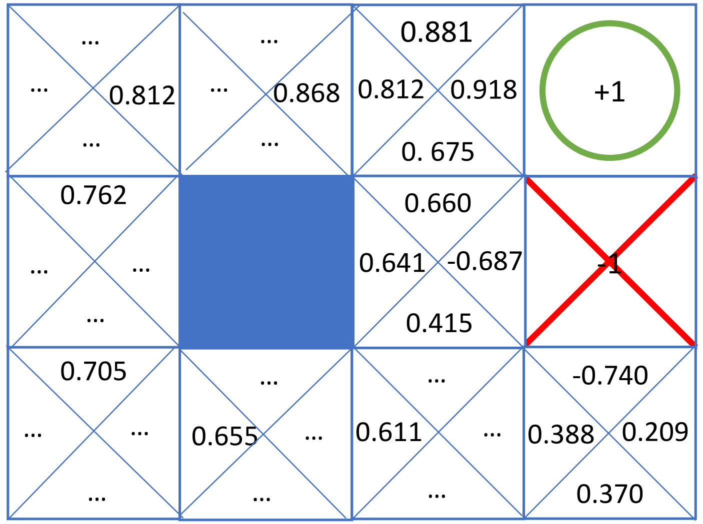
Value of every state V



Optimal policy π^*



Q-value of every action in every state



*Not all values displayed, just example values

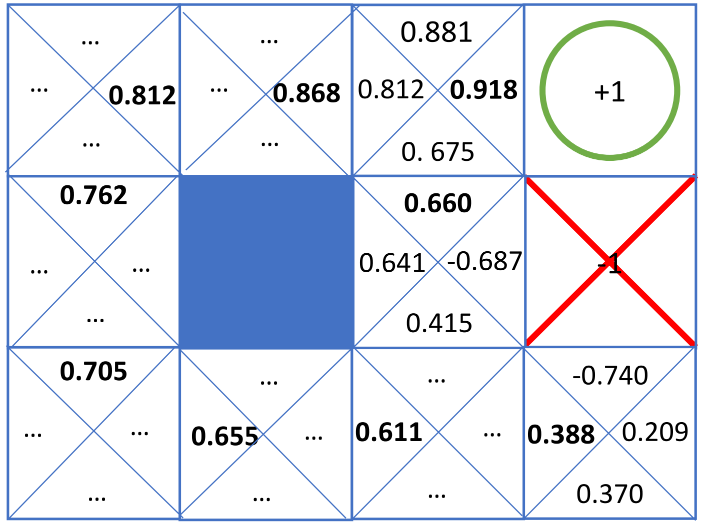
Value of every state V

0.812	0.868	0.918	+1
0.762		0.660	-1
0.705	0.655	0.611	0.388

Optimal policy π^*

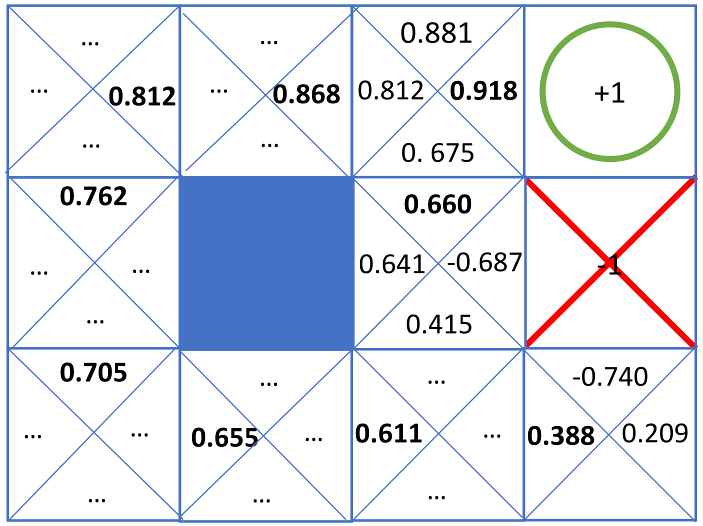
→	→	→	+1
↑		↑	-1
↑	←	←	←

Q-value of every action in every state

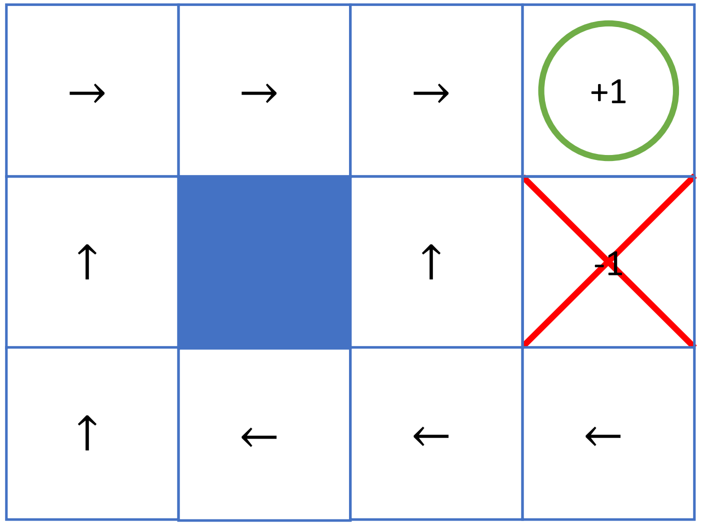


*Not all values displayed, just example values

Q-value of every action in every state



Optimal policy π^*



If we knew the Q-value of every action at every state, then finding the optimal policy is straightforward

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

And finding the V-value of a state is also straightforward

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

The optimal policy is guided by the Q values
i.e. **optimal policy = “follow the Q-values”**

Relationship between V, Q, π^*

$$V(s) = Q(s, \pi^*(s))$$

Bellman equation for the Q-value

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

Bellman equation for the Q-value

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

$$Q(s,a) = \mathbb{E}_{s'}\left[R(s) + \gamma \max_{a'} Q(s',a') \,|s,a\right]$$

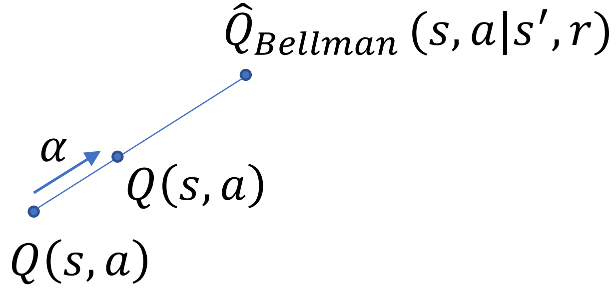
Keep in mind: policy ≡ Q-value
$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$

Q-learning (for finding the optimal policy of an MDP) with learning rate α

- Initialize $Q(s,a)$ at some random values at all states s and actions a (*i.e.* initialize random policy)
- Start in state s_0
- Set current state s= s_0
  - From current state s, choose action a by picking one of these two choices (ϵ-greedy strategy):
    - [Exploitation, Being greedy] Either by following the current policy $\operatorname{argmax}_a Q(s,a)$
    - [Exploration, with probability ϵ] Or by picking a completely random action a
  - Execute a
  - Observe the landed state s', and the obtained reward r (we have now collected an experience record data point $\langle s, a, s', r\rangle$)
  - From this observation, update value of $Q(s,a)$ by taking a stochastic gradient step towards

$$\hat{Q}_{Bellman}(s,a|s',r) = r + \gamma \max_{a'} Q(s',a')$$

$$Q(s,a) = Q(s,a) + \alpha[\hat{Q}_{Bellman}(s,a|s',r) - Q(s,a)]$$



  - Update current state s=s'
  - Loop
- Until convergence of Q/convergence of π^*

This approach is called **Tabular Q-learning**, which means it tries to build a table of Q-values for every (state,action) pair

Problematic with continuous state spaces or continuous action spaces

even with discretization and finite state space: huge number of states (Tetris has 10^{60} states $\times$ 3 actions)

Solution: use function approximation with parametric model

instead of learning $Q(s, a)$ for every (s, a), **parameterize $\boldsymbol{Q}$ as $\boldsymbol{Q_\theta}$** (for example linear model, neural network), and **learn the parameter $\boldsymbol{\theta}$** from the observations, this is called **Approximate Q-learning**

when Q_θ is a deep learning model (for example a CNN, taking the raw pixels of the game as the state of the game), then we talk about **Deep Reinforcement Learning**

Deep RL = Q-value of every action as a deep-learning regression model, called the Q-network

Note that it is different from a supervised learning problem as a classification problem on the actions from the observation of the actions taken by human agents. Here there is no human agent, the agent generates the data it needs and learns a Q-value function

Keep in mind: policy ≡ Q-value
$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q_\theta(s,a)$

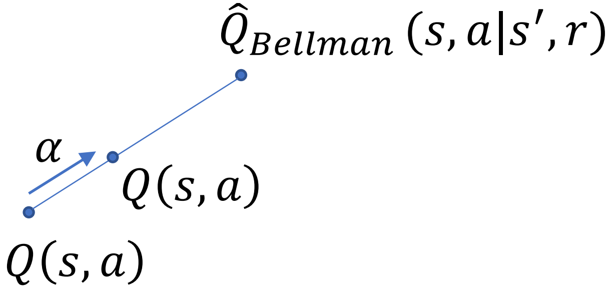
Approximate Q-learning algorithm (for finding the optimal policy of an MDP) with learning rate α

- Initialize θ at some random values (*i.e.* initialize random policy)
- Start in state s_0
- Set current state s= s_0
    - From current state s, choose action a by picking one of these two choices:
        - [Exploitation] Either by following the current policy $\operatorname{argmax}_a Q_\theta(s,a)$
        - [Exploration] Or by picking a completely random action a

    - Execute/simulate a
    - Observe the landed state s', and the obtained reward r (we have now collected an experience record data point $\langle s, a, s', r\rangle$)
    - From this observation, update value of $Q_\theta(s,a)$ by taking a stochastic gradient step towards

$$\hat{Q}_{Bellman}(s,a|s',r) = r + \gamma \max_{a'} Q_\theta(s',a')$$

$$\theta = \theta - \alpha \frac{\partial}{\partial\theta}\left[\hat{Q}_{Bellman}(s,a|s',r) - Q_\theta(s,a)\right]^2\Big|_\theta$$



    - Update current state s=s'
    - Loop
- Until convergence of Q/convergence of π^*

Keep in mind: policy ≡ Q-value
$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q_\theta(s,a)$

Approximate Q-learning algorithm (for finding the optimal policy of an MDP) with learning rate α

- Initialize θ at some random values (*i.e.* initialize random policy)
- Start in state s_0
- Set current state s= s_0
  - From current state s, choose action a by picking one of these two choices:
    - [Exploitation] Either by following the current policy $\operatorname{argmax}_a Q_\theta(s,a)$
    - [Exploration] Or by picking a completely random action a
  - Execute/simulate a
  - Observe the landed state s', and the obtained reward r (we have now collected an experience record data point $\langle s, a, s', r\rangle$)
  - From this observation, update value of $Q_\theta(s,a)$ by taking a stochastic gradient step towards

$$\hat{Q}_{Bellman}(s, a|s', r) = r + \gamma \max_{a'} Q_\theta(s', a')$$

Chasing a moving target

$$\theta = \theta - \alpha \frac{\partial}{\partial\theta}\left[\hat{Q}_{Bellman}(s, a|s', r) - Q_\theta(s,a)\right]^2\Big|_\theta$$

Failed iid assumption for SGD

  - Update current state s=s'
  - Loop
- Until convergence of Q/convergence of π^*

Two problems with this naïve approach

To stabilize Approximate Q-learning, Minh et al, 2015, introduce two improvements:

- **Experience replay**, store 1M transitions (experience data point) in memory buffer, then sample minibatches from those for SGD, don't use current current transition for SGD, store it in memory buffer
- Use **target network** to compute the target of Q, update target network with Q-network every 10000 iterations

DQN algorithm with experience replay (for finding the optimal policy of an MDP) with learning rate α

- Initialize θ (*Q-network*) at some random values (*i.e.* initialize random policy), initialize θ^- to θ (θ^- is the *target network, target network=Q-network at initialization*)
- Start in state s_0
- Set current state s= s_0
  - From current state s, choose action a by picking one of these two choices:
    - [Exploitation] Either by following the current policy $\operatorname{argmax}_a Q_\theta(s,a)$
    - [Exploration] Or by picking a completely random action a
  - Execute/simulate a
  - Observe the landed state s', and the obtained reward r (we have now collected an experience record data point$\langle s,a,s',r\rangle$)
  - Store $\langle s,a,s',r\rangle$ in *replay buffer* $\mathcal{D}$ (buffer capacity 1M, FIFO)
  - Sample minibatches of 32 tuples $\langle s_i,a_i,s_i',r_i\rangle$ of iid from $\mathcal{D}$ to perform SGD
  - [Update Q-network only, not target network] On that minibatch, update value of $Q_\theta(s_i,a_i)$ by taking a stochastic gradient step towards

$$\hat{Q}_{Bellman,i}(s_i,a_i\,|s_i',r_i) = r_i + \gamma \max_{a'} Q_{\theta^-}(s_i',a')$$

$$\theta = \theta - \alpha \frac{\partial}{\partial\theta} \mathbb{E}_{\langle s_i,a_i,s_i',r_i\rangle}\left[\hat{Q}_{Bellman,i}(s_i,a_i|s_i',r_i) - Q_\theta(s_i,a_i)\right]^2\Big|_\theta$$

  - Every 10000 iteration reset θ^- to θ (reset target network to Q-network)
  - Update current state s=s'
  - Loop
- Until convergence of Q/convergence of π^*