

Reinforcement Learning

Deep Reinforcement Learning

Mathieu Goutay, Nokia Bell Labs

mathieu.goutay@nokia.com

INSA Lyon, France

Follow the course at

mgoutay.github.io

→ Blog → Machine Learning Course

(Deep) Reinforcement Learning

Lesson

Slides are available [here](#)

1. Reinforcement Learning, part I

1. Multi-armed Bandits
2. Action, reward, action-value, estimated action-value
3. Policies
4. Your turn 😊

2. Reinforcement Learning, part II

1. Classic RL problem and Markov Decision Process
2. Return, state-value, action-value
3. Temporal Difference Learning
4. Your turn 😊

3. Deep Reinforcement Learning

1. Q-network
2. Experience replay
3. Target Network
4. Your turn 😊

Reinforcement Learning, Part I

Multi-armed Bandit

Problem formulation:

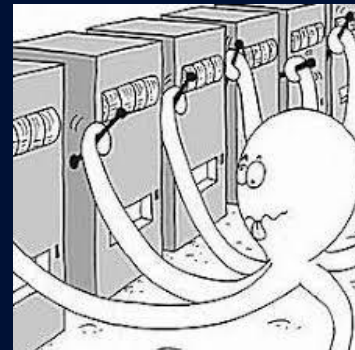
- You have a Wi-Fi network with 7 different channels
 - You need to transmit 10.000 packets
- Which channel do you choose?

Framework:

- For each attempt, you choose a channel (**action**)
- If the packet was sent, you got a **reward** : 1
- If there was a collision, you got no **reward** : 0

Goal: **Online training**

→ Find the best action to maximize the total number of transmitted packets (received rewards)



Reinforcement Learning, Part I

Action, reward, action-value, estimated action-value

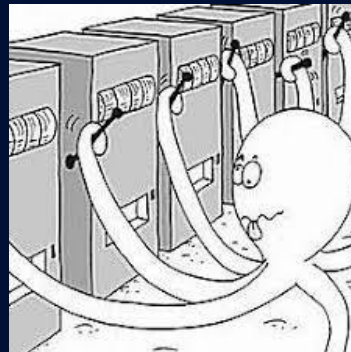
Notations:

- The action and reward at times step t denoted in capital letters: A_t, R_t
- Their possible values are denoted in lower case : a, r

We evaluate each action with its **action-value function** :

- What is the expected return if we choose this channel (action)?

$$q_*(a) = E[R_t | A_t = a]$$



Problem : We don't know it ☹

Reinforcement Learning, Part I

Action, reward, action-value, estimated action-value

We need to **estimate the action-value function** :

- Test each channel many times
- Compute an average for each channel

$$Q_t(a) = \frac{\text{sum of rewards when we took the action } a}{\text{number of time we took the action } a}$$
$$= \frac{\sum_{i=1}^{N_t(a)} R_i^a}{N_t(a)}$$

Notations:

- The true action-value function is denoted in lower case : $q_*(a)$
- Its estimate at time t is denoted in capital letters : $Q_t(a)$
- $N_t(a)$ denotes the number of times action a has been selected at time t
- $[R_1^a, R_2^a, \dots, R_{N_t(a)}^a]$ denotes the rewards we got when taking the action a

Reinforcement Learning, Part I

Action, reward, action-value, estimated action-value

We can then choose the best action at time t :

$$A_t = \arg \max_a Q_t(a)$$

We chose the action that will most probably give the best reward

Tradeoff:

- We need to test each channel many times to have the best estimate $Q_t(a) \rightarrow$ Exploration
- We need to take the best action to maximize the rewards \rightarrow Exploitation

Reinforcement Learning, Part I

Policies

How to both explore and exploit ? We follow a **policy π**

- The policy define how we chose the action at each time step t

The most basic policy is the **ϵ -greedy policy**:

At each time step t :

- With a probability ϵ , take a random action
 - Refine the action-value estimation for that action
- With a probability $(1 - \epsilon)$, take the best action $A_t = \arg \max_a Q_t(a)$
 - Maximize the reward

ϵ defines the tradeoff exploration/exploitation

Reinforcement Learning, Part I

Policies

Another one is the **Upper-Confidence-Bound (UCB)** policy:

$N_t(a)$ denotes the number of times action a has been selected at time t
Select the best action according to :

$$A_t = \arg \max_a \left[Q_t(a) + \sqrt{\frac{\ln(t)}{N_t(a)}} \right]$$

←
Estimate of the average return
for the action a

←
Measure of uncertainty
of the estimate $Q_t(a)$

Often takes the best action but still refine the estimate $Q_t(a)$

Let's play 😊

Exercise

Multiple Access Channel with Reinforcement Learning

1. Reinforcement Learning, part I

1. Multi-armed Bandits
2. Action, reward, action-value, estimated action-value
3. Policies
4. Your turn 😊

2. Reinforcement Learning, part II

1. Classic RL problem and Markov Decision Process
2. Return, state-value, action-value
3. Temporal Difference Learning
4. Your turn 😊

3. Deep Reinforcement Learning

1. Q-network
2. Experience replay
3. Target Network
4. Your turn 😊

Reinforcement Learning, Part II

Classic RL problem

Problem formulation:

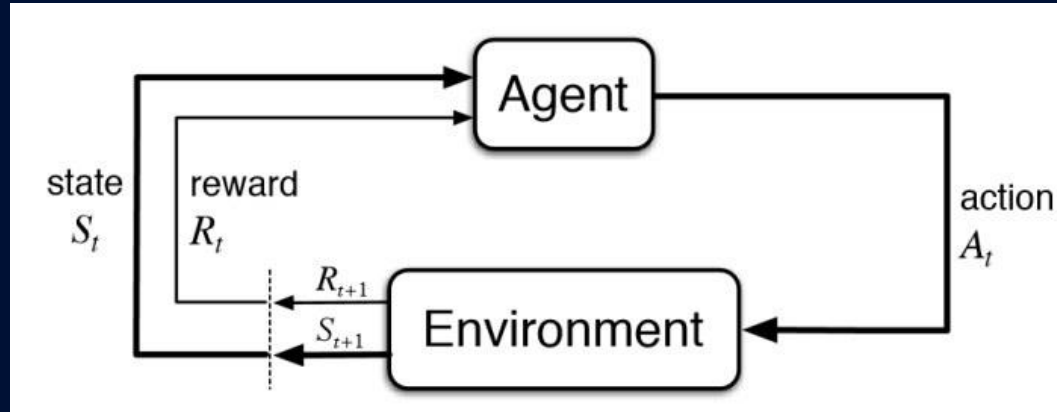
- You are in an industrial environment with 7 different channels
 - All existing machines transmit with a certain periodicity
 - You install a new IoT sensor, and need to find on which channel to transmit 5 packets
 - You can sense the channels before sending a packet(!)
- Which channels do you choose?

Framework: At each time step

- You sense the channels (**state**)
- You see if the previously transmitted packet has been correctly received (**reward**)
- You decide on which channel you transmit next (**action**)

Reinforcement Learning, Part II

Classic RL problem



The IoT sensor is a **agent** which interacts with an **environment**

At each time step t :

- The agent receives a state (sensed channels) and a reward (previous tx successful?)
- The agent takes an action (choose the next channel)

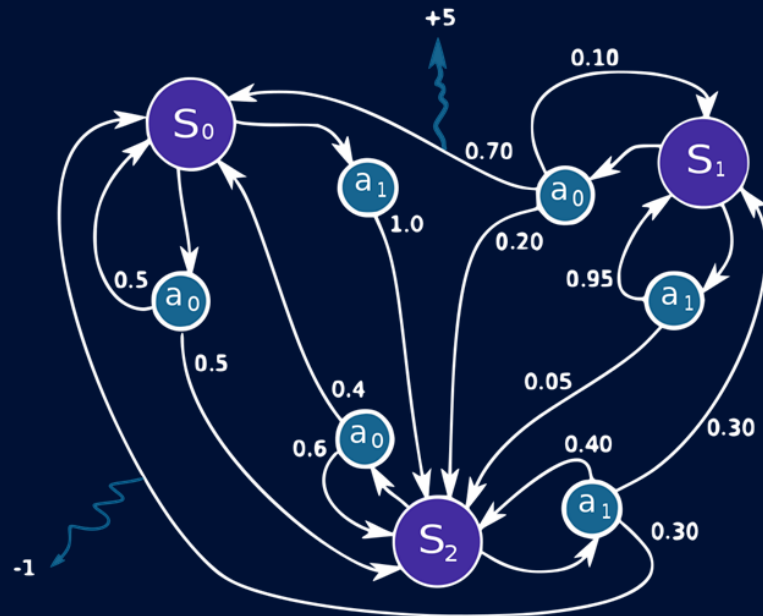
The sequence is defined by $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$

Reinforcement Learning, Part II

Markov Decision Process

A **Markov Decision Process** (MDP) is a 4-tuple $(\mathcal{S}, \mathcal{A}_s, \mathcal{P}_a, \mathcal{R}_a)$:

- \mathcal{S} is a finite set of states
- \mathcal{A}_s is the finite set of actions available from state s
- $\mathcal{P}_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will lead to state s' at time $t + 1$
- $\mathcal{R}_a(s, s')$ is the reward received after transitioning from state s to state s' , due to action a



Reinforcement Learning, Part II

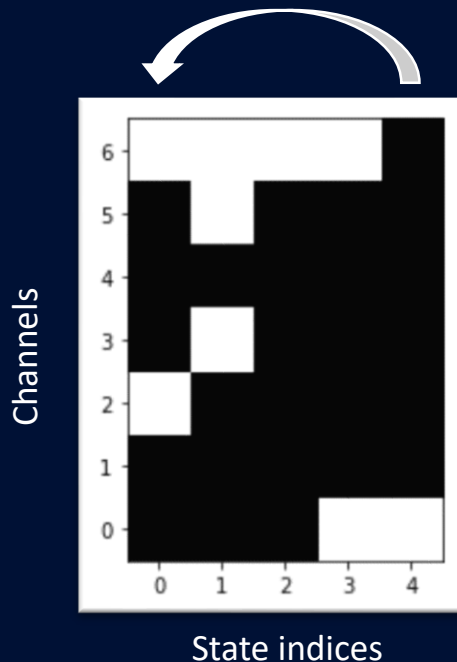
Classic RL problem

For simplicity, let's have a known deterministic environment:

- The white boxes are free channels
- The black boxes are already used channels
- You have 5 different channel states
- You start at the “starting point”

Goal : Find the best channels for the 5 transmissions

The transmission of 5 packets is called an **episode**



Reinforcement Learning, Part II

Classic RL problem

The **state** consists of:

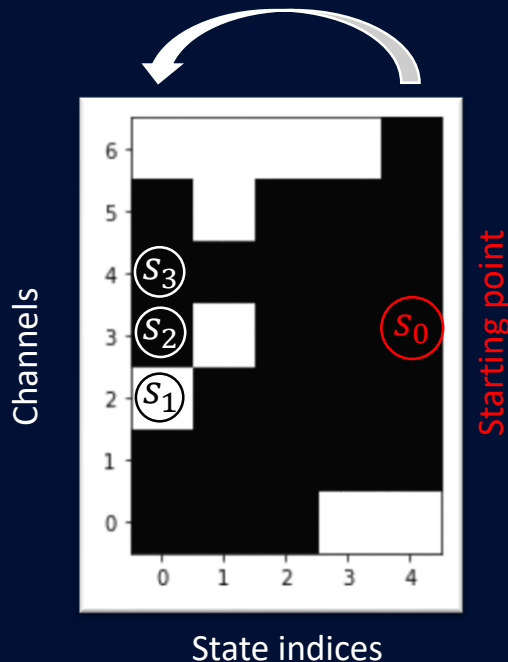
- The state index we sense (5 possibilities)
- The channel in which we transmitted (7 possibilities)
- A Boolean to indicate if we reached the last state (5 transmissions)

The **rewards** are :

- 1 if the transmission was successful
- 0 otherwise

The possible **actions** are:

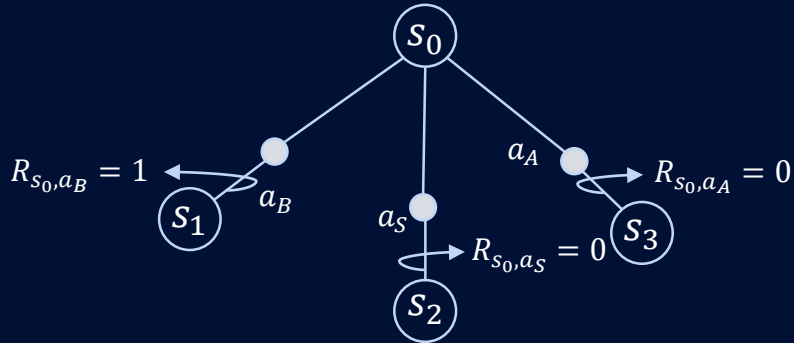
- Transmit in the channel above (mod 7) : A
- Transmit in the same channel : S
- Transmit in the channel below (mod 7) : B



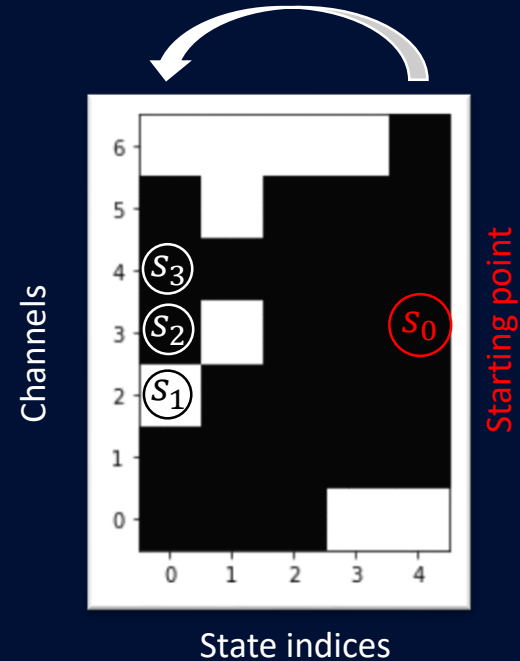
Reinforcement Learning, Part II

Classic RL problem

First state : $S_0 = [state_index, prev_channel, bool] = [4, 3, 0]$



Possible second state : $S_1 = [0, 2, 0]$, $S_2 = [0, 3, 0]$, $S_3 = [0, 4, 0]$

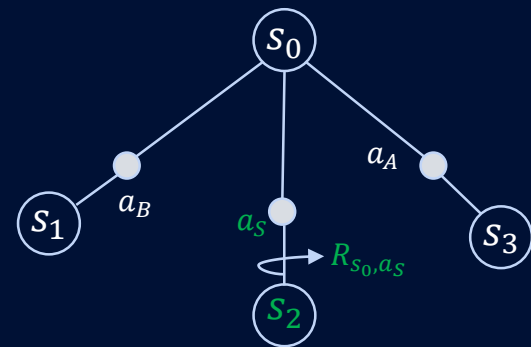


Reinforcement Learning, Part II

Return, state-value, action-value

The **discounted return** is the sum of rewards after a time step t :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$
$$= R_{t+1} + \gamma G_{t+1}$$



The **discount factor** γ , with $0 \leq \gamma \leq 1$, is used for:

- Having a finite return even if the number of future time steps k is infinite
- Maximizing short-term ($\gamma = 0$) or long-term ($\gamma = 1$) reward

Note that in our problem, the maximum time step is 5

Reinforcement Learning, Part II

Return, state-value, action-value

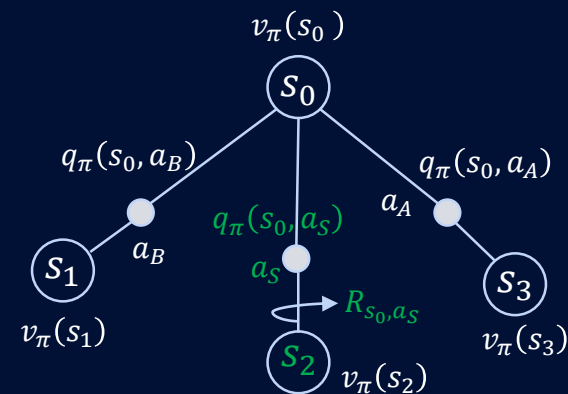
For a given state s :

- The **policy $\pi(a|s)$** is the probability of choosing an action a
- The **state-value function** is the expected return

$$v_{\pi}(s) = E[G_t(\pi) | S_t = s]$$

- The **action-value function** is the expected return if we choose action a

$$q_{\pi}(s, a) = E[G_t(\pi) | S_t = s, A_t = a]$$



We want to find a policy that choose the best actions according to the $q_{\pi}(s, a_i)$

But how do we estimate those $q_{\pi}(s, a_i)$?

Reinforcement Learning, Part II

Return, state-value, action-value

The optimal policy is the one that maximizes $v_\pi(s)$ and $q_\pi(s)$:

- The optimal action-value function is

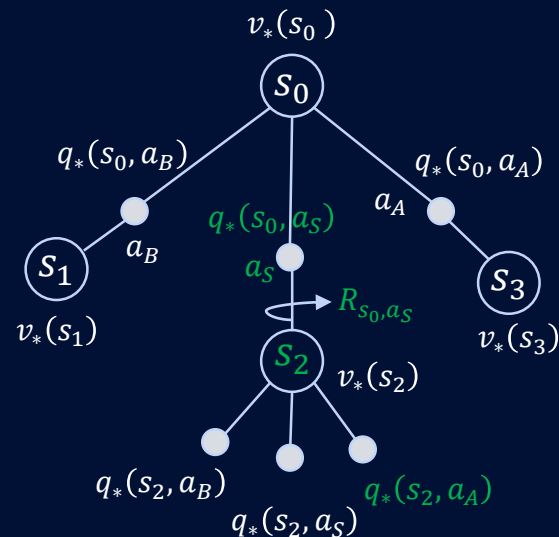
$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- The optimal state-value function is

$$\begin{aligned} v_*(s) &= \max_{\pi} v_{\pi}(s) \\ &= \max_a q_*(s, a) \end{aligned}$$

Bellman optimality equation :

$$\begin{aligned} q_*(s, a) &= \max_{\pi} \mathbb{E}[G_t(\pi) | S_t = s, A_t = a] \\ &= \max_{\pi} \mathbb{E}[R_{t+1} + \gamma G_{t+1}(\pi) | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \mathbb{E} \left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a \right] \end{aligned}$$



Reinforcement Learning, Part II

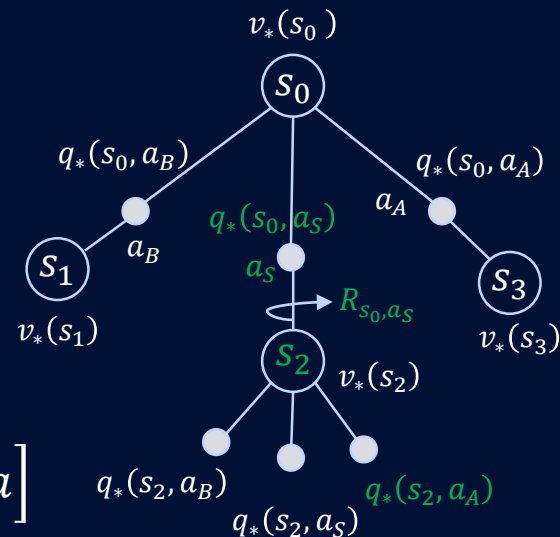
Return, state-value, action-value

Bellman optimality equation

Trick : when the latest reward is received, there is no more state
“End” Boolean ξ : 1 if we reached the last state, 0 otherwise

$$q_*(s, a) = \mathbb{E} \left[R_{t+1} + \gamma(1 - \xi) \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right]$$

Next goal : find an estimate $Q_*(s, a)$ of $q_*(s, a)$



Reinforcement Learning, Part II

Temporal Difference Learning

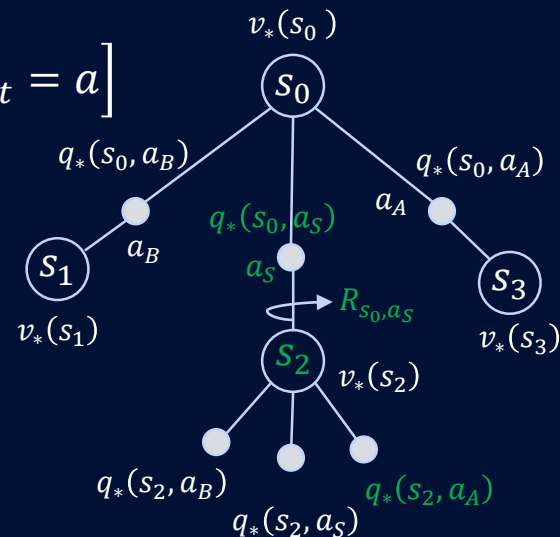
Estimate $q_*(s, a) = E \left[R_{t+1} + \gamma(1 - \xi) \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a \right]$

Q-Learning: wait to finish a $S_t, A_t, R_{t+1}, S_{t+1}$ (with a policy π)

(0. Initialize all $Q_*(s, a)$ randomly)

1. Take the $Q_*(S_t, A_t)$ associated with your state and action
2. When in S_{t+1} , take the best q-value: $\max_a Q_*(S_{t+1}, a)$
3. Compute a better estimate $Q'_*(S_t, A_t) = R_{t+1} + \gamma(1 - \xi) \max_a Q_*(S_{t+1}, a)$
4. Compute an error $Q'_*(S_t, A_t) - Q_*(S_t, A_t)$
5. Update $Q_*(S_t, A_t) \leftarrow Q_*(S_t, A_t) + \alpha [Q'_*(S_t, A_t) - Q_*(S_t, A_t)]$

α is the learning rate



Temporal Difference (TD): use the time step $t+1$ to refine the time step t

Reinforcement Learning, Part II

Temporal Difference Learning

With a policy π , record $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$

- SARSA: **On-policy** method

$$Q_{\pi}(S_t, A_t) \leftarrow Q_{\pi}(S_t, A_t) + \alpha [R_{t+1} + \gamma(1 - \xi) Q_{\pi}(S_{t+1}, A_{t+1}) - Q_{\pi}(S_t, A_t)]$$



Behavior policy
i.e., ϵ -greedy



Behavior policy
i.e., ϵ -greedy

You estimate the Q_{π} according to the policy you are using

- Q-Learning : **Off-policy** method

$$Q_*(S_t, A_t) \leftarrow Q_*(S_t, A_t) + \alpha \left[R_{t+1} + \gamma(1 - \xi) \max_a Q_*(S_{t+1}, a) - Q_*(S_t, A_t) \right]$$

Optimal policy

Optimal policy

You use the policy π to explore and estimate the optimal policy

Reinforcement Learning, Part II

Classic RL problem

The **state** comprises both:

- The channel in which we transmitted (7 possibilities)
- The channel state index we sense (5 possibilities)
- The ξ Boolean (=1 if it's the end, =0 otherwise)

The **rewards** are :

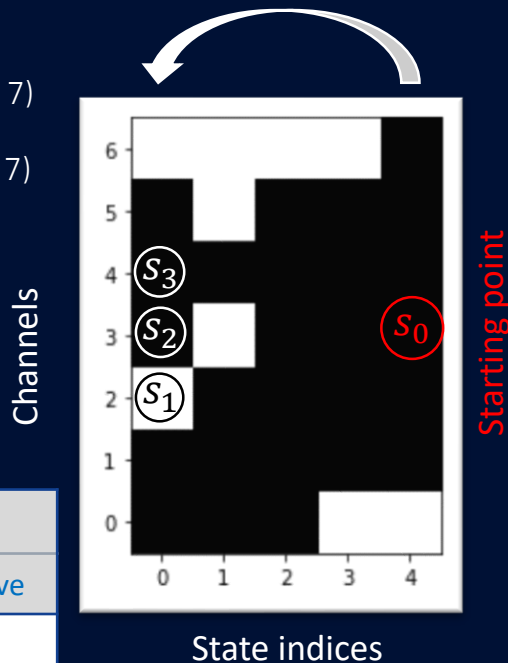
- 1 if the transmission was successful
- 0 otherwise

Q-Learning : fill the **Q-table** following a ϵ -greedy policy

	State index 0			...	State index 4		
	a_B : Below	a_S : Same	a_A : Above	...	a_B : Below	a_S : Stay	a_A : Above
...
Ch 3	$Q_*([3, 0], a_B)$	$Q_*([3, 0], a_S)$	$Q_*([3, 0], a_A)$...	$Q_*([3, 4], a_B)$	$Q_*([3, 4], a_S)$	$Q_*([3, 4], a_A)$
...

The possible **actions** are :

- Tx in channel above (mod 7)
- Tx in same channel
- Tx in channel below (mod 7)



Let's play 😊

Exercise

Multiple Access Channel with Reinforcement Learning

1. Reinforcement Learning, part I

1. Multi-armed Bandits
2. Action, reward, action-value, estimated action-value
3. Policies
4. Your turn 😊

2. Reinforcement Learning, part II

1. Classic RL problem and Markov Decision Process
2. Return, state-value, action-value
3. Temporal Difference Learning
4. Your turn 😊

3. Deep Reinforcement Learning

1. Q-network
2. Experience replay
3. Target Network
4. Your turn 😊

Deep Reinforcement Learning

Q-Network

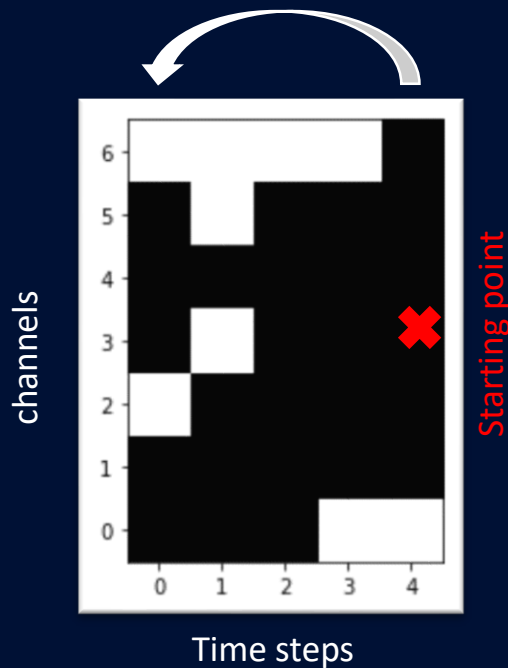
Previously, we knew that there were only 5 channel indices

→ What if we don't know that?

One state is now defined by the state vector :



And the ξ boolean



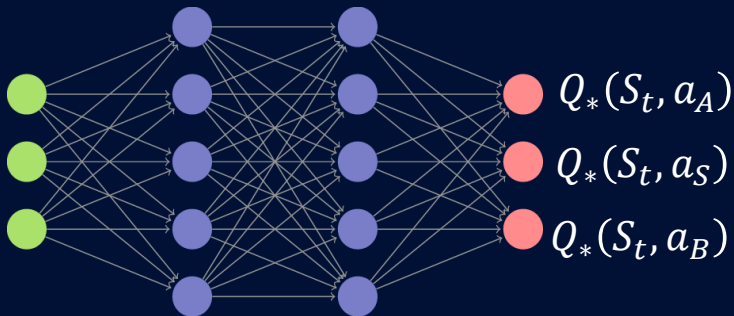
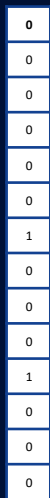
Deep Reinforcement Learning

Q-Network

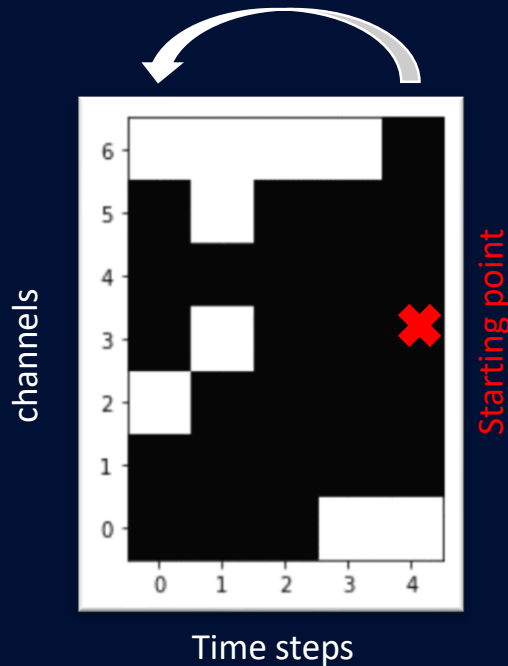
We don't want to store a huge Q-table.

We can use a **Q-Network** instead :

S_t vector



The Q-Network outputs all the Q values for a given states



Deep Reinforcement Learning

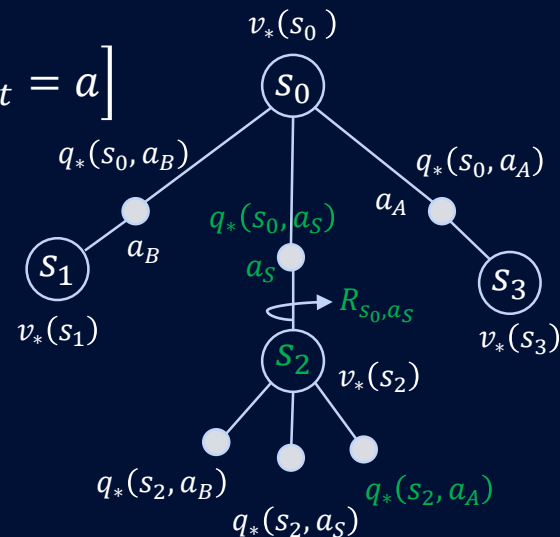
Q-Network

Estimate $q_*(s, a) = E \left[R_{t+1} + \gamma(1 - \xi) \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a \right]$

Q-Learning : wait to finish a $S_t, A_t, R_{t+1}, S_{t+1}$ (with a policy π)

(0. Initialize Q-Network randomly)

1. Take the $Q_*(S_t, A_t)$ associated with your state and action
2. When in S_{t+1} , take the best q-value : $\max_a Q_*(S_{t+1}, a)$
3. Compute a better estimate $Q'_*(S_t, A_t) = R_{t+1} + \gamma(1 - \xi) \max_a Q_*(S_{t+1}, a)$
4. Compute loss : $MSE(Q'_*(S_t, A_t), Q_*(S_t, A_t))$
5. Update parameters of the Q-Network by SGD to minimize the loss



Deep Reinforcement Learning

Q-Network

$$\text{loss} = \text{MSE} \left(\overbrace{R_{t+1} + \gamma(1 - \xi) \max_a Q_*(S_{t+1}, a)}^{\text{Target}} - \overbrace{Q_*(S_t, A_t)}^{\text{Prediction}} \right)$$

Each time we update for one prediction, every parameters in the NN changes!
Two problems arises :

- *Correlation* : when you follow the trajectory, your NN will be optimized only for the last few (s, a) that you took
- *Nonstationary target* : each time we update the NN, the target change as well
→ not stable

Deep Reinforcement Learning

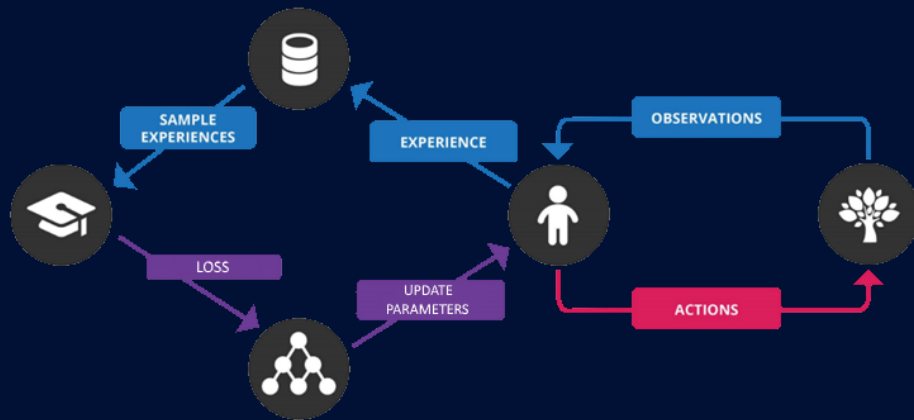
Experience Replay

To remove *correlation*, we store all experiences $(S_t, A_t, R_{t+1}, S_{t+1})$ in a dataset.

Then, at each iteration, we perform **experience replay**:

- We take a random batch of experiences
- We compute the predictions and targets
- We evaluate the loss and update the Q-Network

$$loss = MSE(\text{predictions}, \text{targets})$$



Circular buffer (FIFO)

S_1, A_1, R_2, S_2
S_2, A_2, R_3, S_3
S_3, A_3, R_4, S_4
S_4, A_4, R_5, S_5
...

Deep Reinforcement Learning

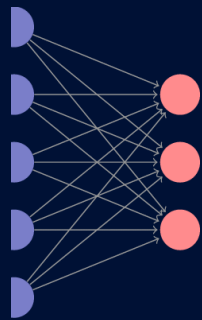
Experience Replay

To remove *correlation*, we store all samples $(S_t, A_t, R_{t+1}, S_{t+1})$ in a dataset.

$$loss = MSE(\text{predictions}, \text{targets})$$

The targets can only reflect the action chosen in the sample

If at state S_t , the action taken was a_A :



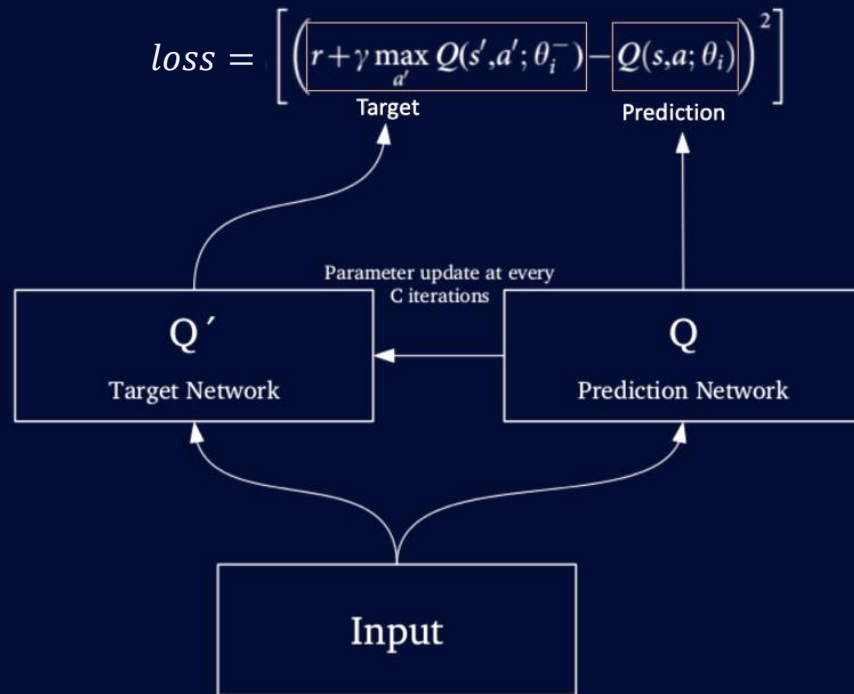
$$MSE \left[\begin{array}{c} \text{Prediction} \\ Q_*(S_t, a_A) \\ Q_*(S_t, a_S) \\ Q_*(S_t, a_B) \end{array} - \begin{array}{c} \text{Target} \\ R_{t+1} + \gamma(1 - \xi) \max_a Q_*(S_{t+1}, a) \\ Q_*(S_t, a_S) \\ Q_*(S_t, a_B) \end{array} \right] = MSE \left[\begin{array}{c} pred - target \\ 0 \\ 0 \end{array} \right]$$

Deep Reinforcement Learning

Target Network

To alleviate the *nonstationary target*, we maintain a **target Q-Network** :

- The targets are computed according to the target network
- The parameters of the target network are updated every C iterations



Target Network

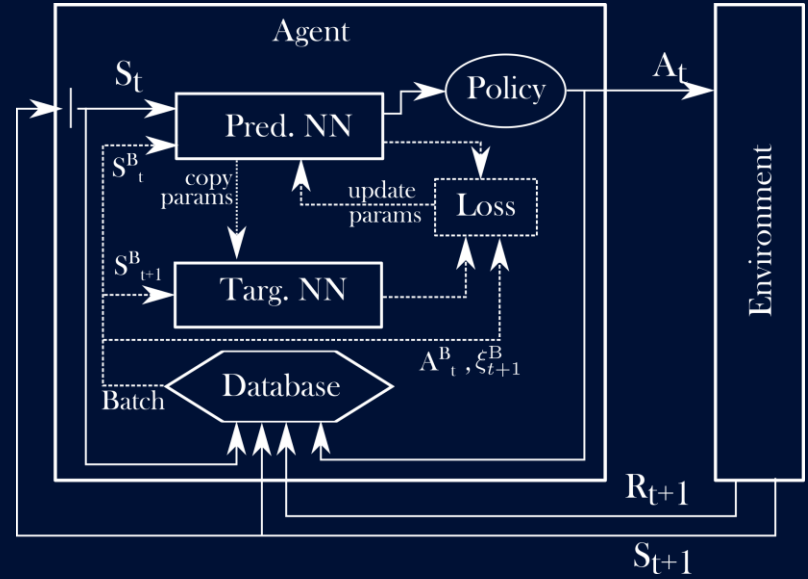
Main algorithm :

Initialize the pred.NN and the targ.NN with the same params

Play actions according to a policy π to populate the datasets

For a given number of episodes :

- While $\xi \neq 1$:
 - Choose an action A_t according to the state S_t and the policy π
 - Receive R_{t+1} , S_{t+1} and store $(S_t, A_t, R_{t+1}, S_{t+1})$ in the database
 - Take a random batch from the database (B = batch size)
 - Compute the loss using targets from the target network
 - Update the prediction Q-Network
- Every C iterations, copy the parameters of the pred.NN to the targ.NN



Let's play 😊

Exercise

Multiple Access Channel with Reinforcement Learning

Thank you

Everything is available on
mgoutay.github.io