KISTI 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

Advanced Topic in Research Data-centric Deep Learning

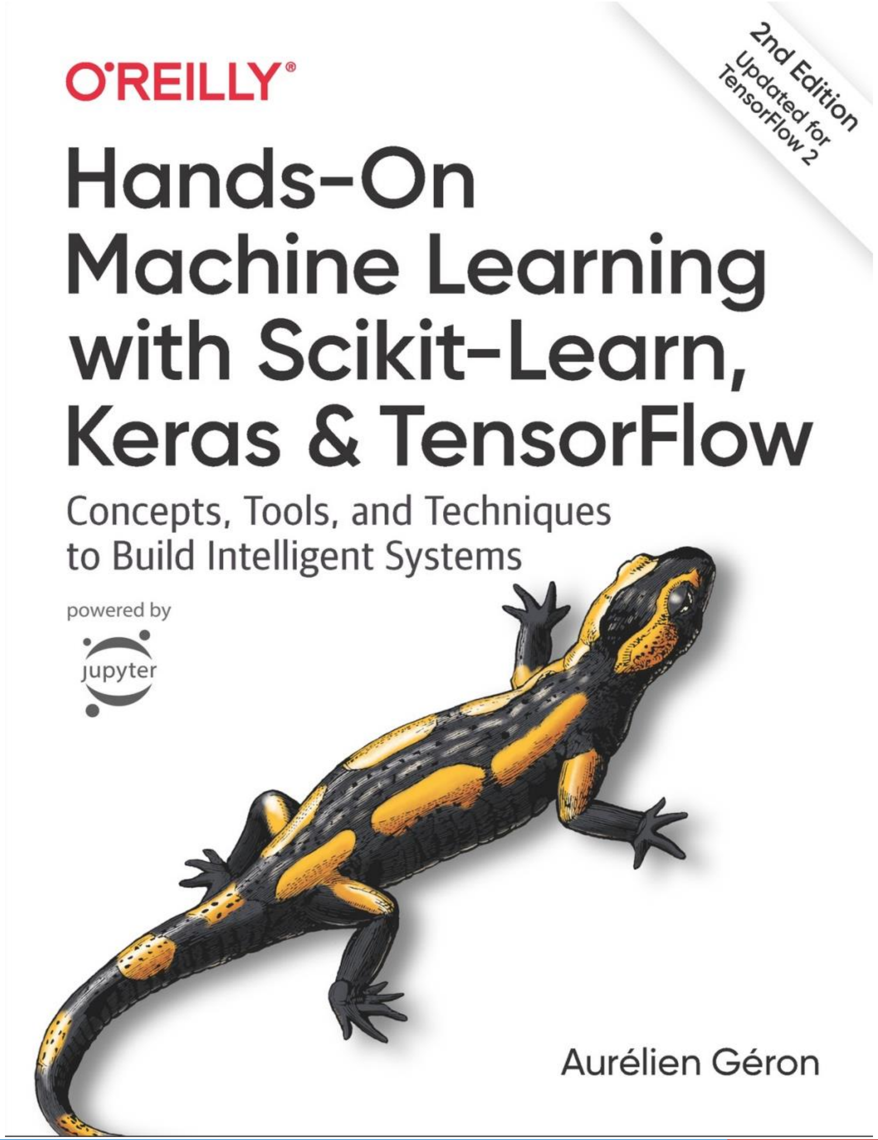
Lec 14: Introduction to Deep Reinforcement Learning

hsyi@kisti.re.kr
Hongsuk Yi (이홍석)

Background material

❖ Background material

- ✓ Reinforcement Learning: An Introduction, Sutton & Barto 2018
  - http://incompleteideas.net/book/the-book-2nd.html
- ✓ Reinforcement Learning Lecture Series 2021
  - https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021
- ✓ https://huggingface.co/blog/deep-rl-intro

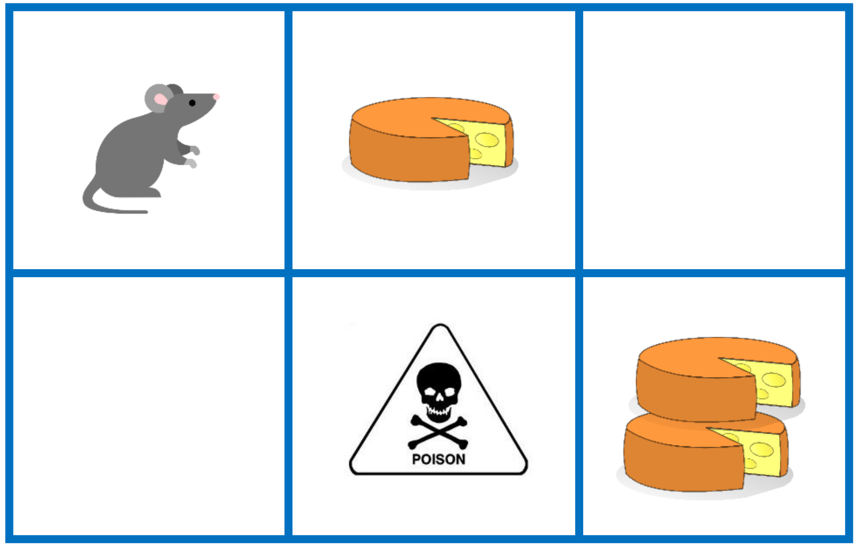


Reviewing the last class:

RL Rule

- ✓ You always start at the same starting point.
- ✓ The goal:
  - eat the big cheese at the bottom right-hand corner and avoid the poison.
- ✓ The epoisoe ends
  - if we eat the poisson, eat the big pile of cheese
  - or if we spent more than 5 steps.
- ✓ The reward function: 0/+1/+10/-10
  - 0: Goting to a state with no cheese
  - +1: with a small cheese
  - +10: with the big pile of cheese
  - -10: with the poison and die
- ✓ Learning rate, Gamma

$$\alpha = 0.1, \gamma = 0.99$$

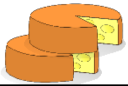


❖ Step 1 : We initialize the Q-Table

Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)

action : a

state : s

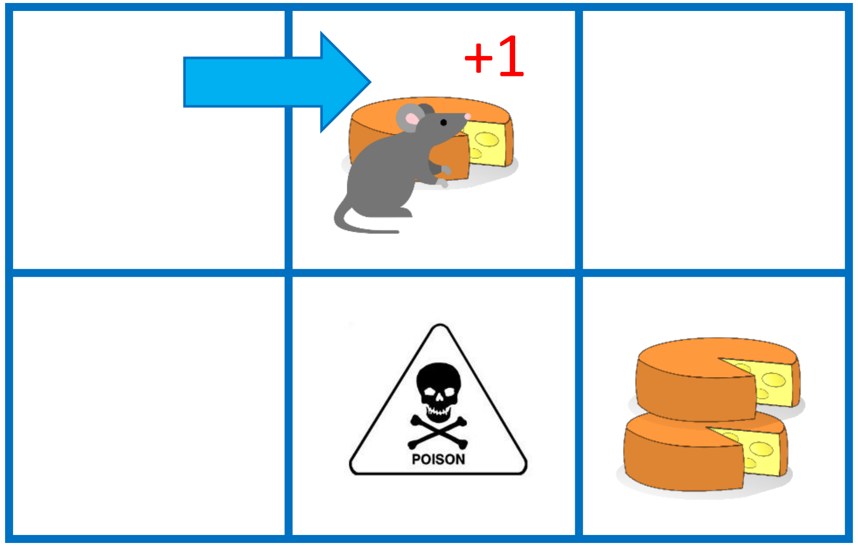
	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

- Step 2 : We took a random action (exploration)

Choose action A_t using policy derived from Q (e.g., ϵ-greedy)

- Step 3 : Observe a reward and a state

Take action A_t and observe R_{t+1}, S_{t+1}



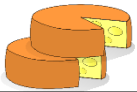
A Q-Learning example

❖ Step 4 : Update ou Q-value estimation

$$\alpha = 0.1, \gamma = 0.99$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

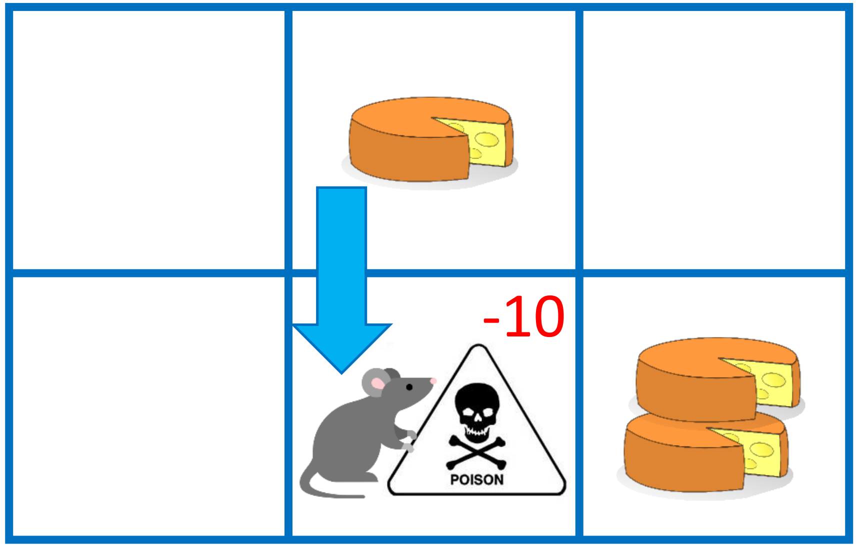
Q(Initial state, Right)=0 +0.1*[1 + 0.99*0 – 0]=0.1

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

A Q-Learning example

❖ Step 5 : A new trial

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$



	←	→	↑	↓
	0	0.1	0	0
	0	0	0	?
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

What is Q-Learning?

- The Q comes from "the Quality" of that action at that state
  - ✓ Q-function contains a Q-table **that has the value of each-state action pair**
- Optimal value function
  - ✓ Finding an optimal value function leads to having an optimal policy

$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

	←	→	↑	↓
	0	0.1	0	0
	0	0	0	?
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

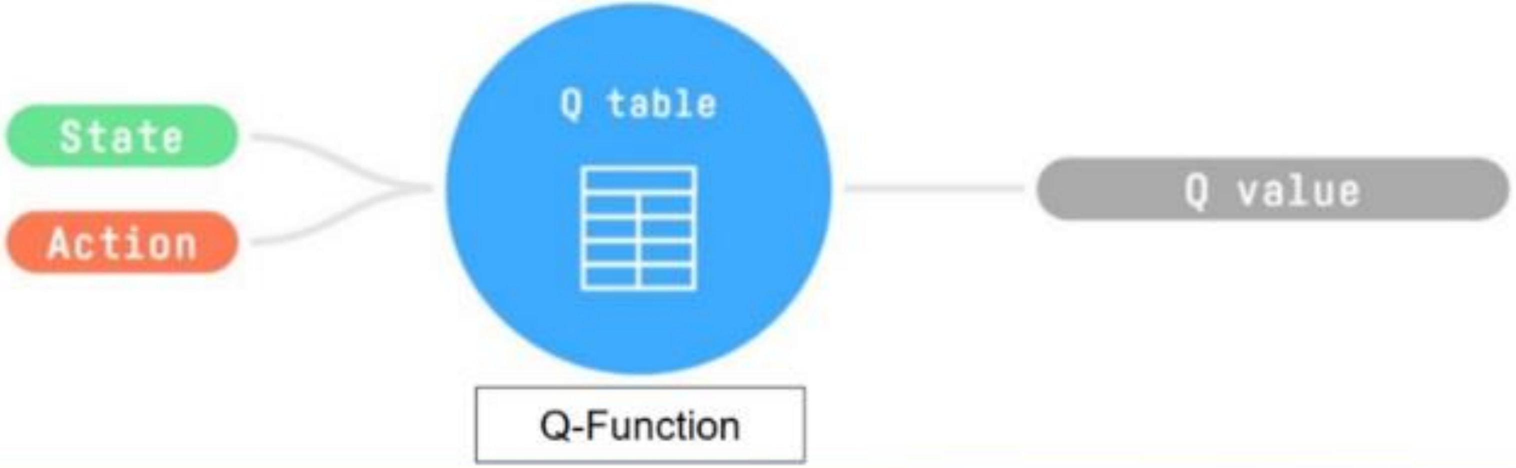
What is Q-Learning?

What is Q-Learning?

- **Q-Learning is an off-policy value-based method**
  - ✓ that uses a Temporal Differnence(TD) approach to train its action-value function
  - ✓ Value-based method
    - finds the optimal policy indirectly by training a value or action-value function that will tell us **the value of each state or each state-action pair.**
  - ✓ Uses a TD approach:
    - **updates its action-value function at each step instead of at the end of the episode.**
  - ✓ Off-policy:
    - using **a different policy for acting and updating using epsilon-greedy policy again**

What is Q-Learning?

- Q-Learning is the algorithm to train an action-value function
  - ✓ that determines the value of being at a particular state and taking a specific action at that state.
    - Given a state and action, Q function outputs a state-action value (also called Q-value)



https://huggingface.co/blog/deep-rl-q-part2

Q-learning: Off-Policy TD Control

KISTI SINCE 1962

- In Q-learning the learned action-value function, Q, directly approximates the optimal action-value function, independent of the policy being followed.
  - the transition probabilities are unknown and the rewards are initially unknown
  - Q-Learning works by watching an agent play (e.g., randomly) and gradually improving its estimates of the Q-Values

$$Q(s_t, a_t) \leftarrow (s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

$$Q(s_t, a_t) \leftarrow \boxed{Q(s_t, a_t)} + \boxed{\alpha} [\boxed{r_{t+1}} + \boxed{\lambda} \boxed{\max_{a} Q(s_{t+1}, a)} - \boxed{Q(s_t, a_t)}]$$

- Current Q-table value we are updating
- Learning rate
- Reward
- Discount
- Estimated reward from our next action

Q-Learning is called an Off-Policy

❖ Off-policy: The Q-learning algorithm

✓ work by "*looking over someone's shoulder.*"

✓ the algorithm attempts to learn about policy π from experience sampled from μ.

✓ the policy being executed is completely random, while the policy being trained will always choose the actions with the highest Q-Values.

- Q-Learning is capable of learning the optimal policy by just watching an agent act randomly
- learning to play golf when your teacher is a drunk monkey

❖ On-policy: The Policy Gradients algorithm, and SARSA

✓ We can say that algorithms classified as **on-policy** are "*learning on the job.*"

- In other words, the algorithm attempts to learn about policy π from experience sampled from π.
- it explores the world using the policy being trained.

The main problem with Q-Learning

❖ It does not scale well to large MDPs with many states and actions

- The number of possible states is greater than $2^{150} \approx 10^{45}$.

❖ Approximate Q-Learning

✓ The solution is to find a function Q(s, a) which approximates the Q-Value of any state-action pair (s, a) using a manageable number of parameters

❖ Deep Q-Learning, DeepMind in 2013

Summary of Q-Learning

State

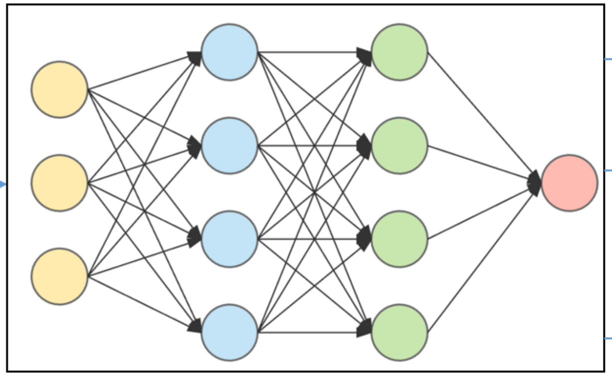
Action

Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q-Value

Q Learning

State



Q-Value Action 1

Q-Value Action 2

-

-

Q-Value Action N

Deep Q Learning

Q-learning Algorithm

KISTI SINCE 1962

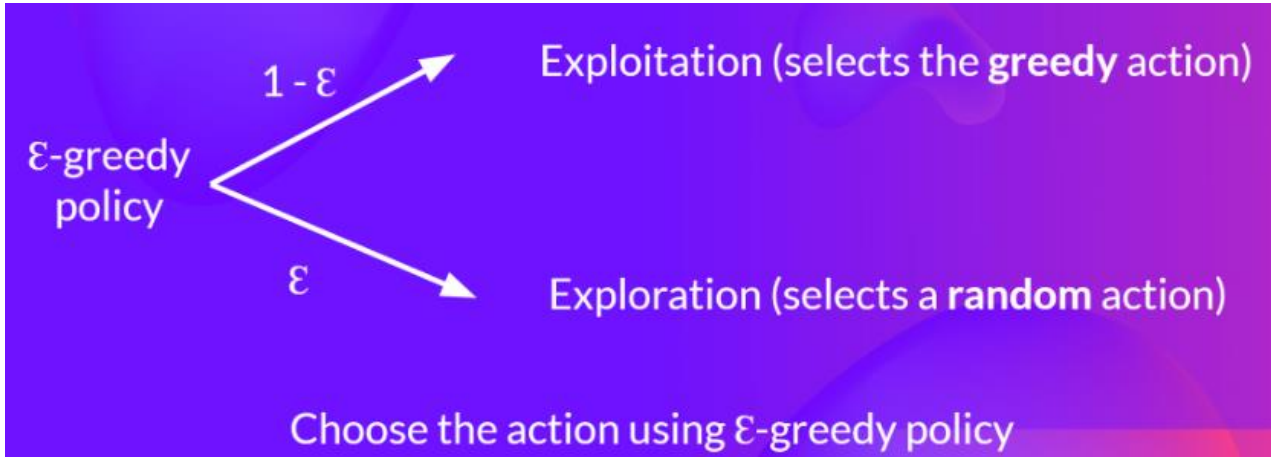
❖ The ε-greedy policy

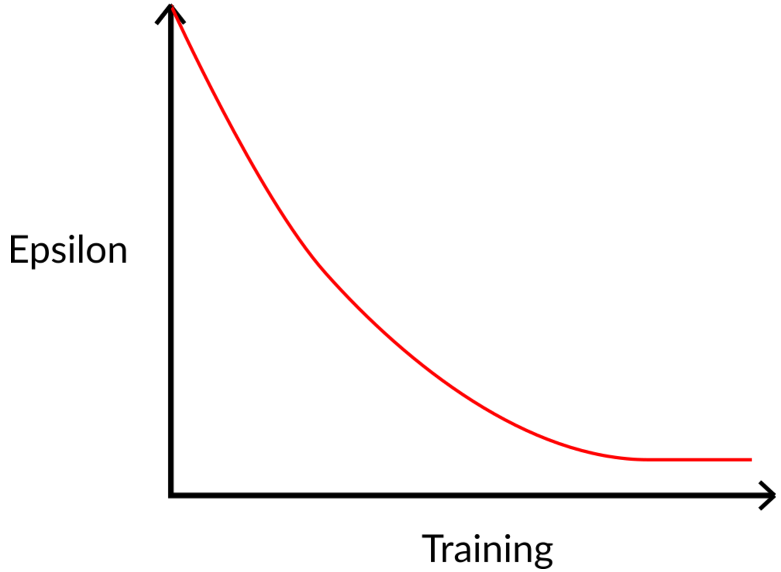
✓ with probability 1 — ε

- we do exploitation : our agent selects the action with the highest state-action pair value

✓ With probability ε:

- we do exploration : trying random action and then gradually reduce it





Q-learning Algorithm

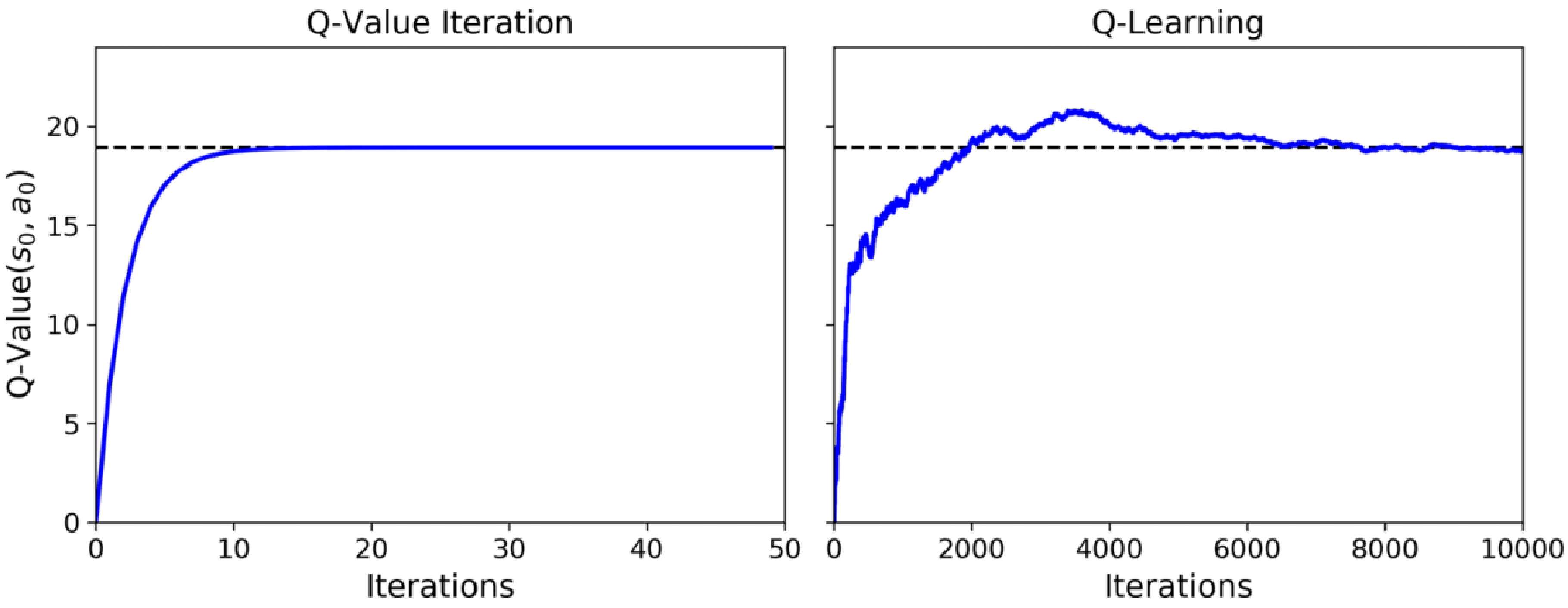
- Step 1: We initialize the Q-Table
- Step 2: Choose action using Epsilon Greedy Strategy
- Step 3: Perform action At, gets reward Rt+1 and next state St+1
- Step 4: Update Q(St, At)

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Q-Learning

- Obviously, not knowing the transition probabilities or the rewards makes finding the optimal policy significantly harder!
  - The Q-Value Iteration algorithm (left) converges very quickly, in fewer than 20 iterations,
  - while the Q-Learning algorithm (right) takes about 8,000 iterations to converge



Deep Q-Network

From Q-Learning to Deep Q-Learning

❖ We learned that Q-Learning is an algorithm we use to train our Q-Function:

✓ an **action-value function** that determines the value of being at a particular state and taking a specific action at that state.

❖ The Q comes from "the Quality" of that action at that state.

✓ Internally, our Q-function has **a Q-table, a table where each cell corresponds to a state-action pair value.**

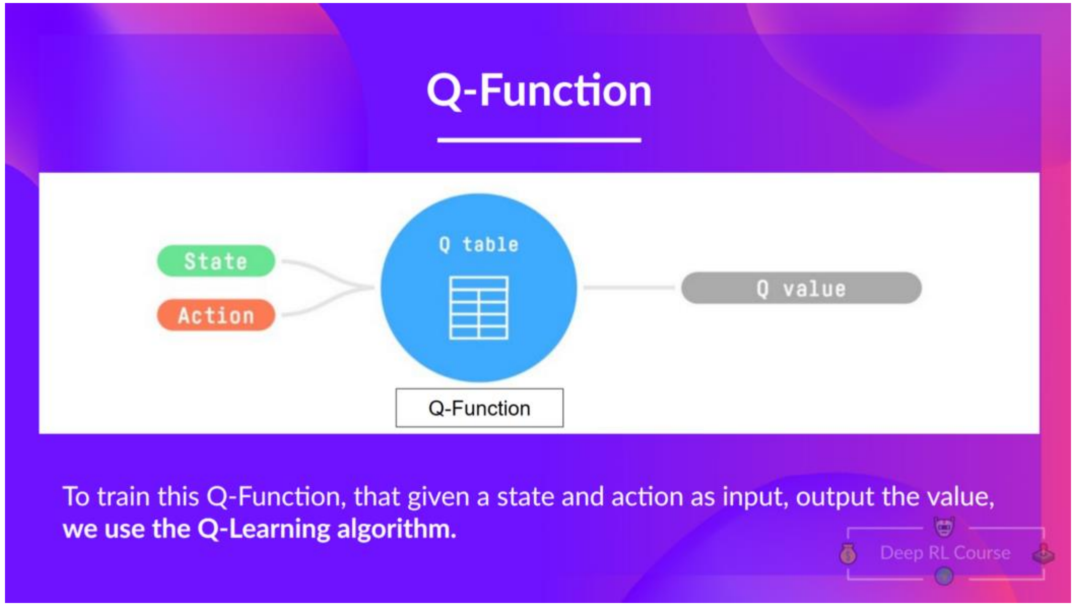
- Think of this Q-table as **the memory or cheat sheet of our Q-function.**

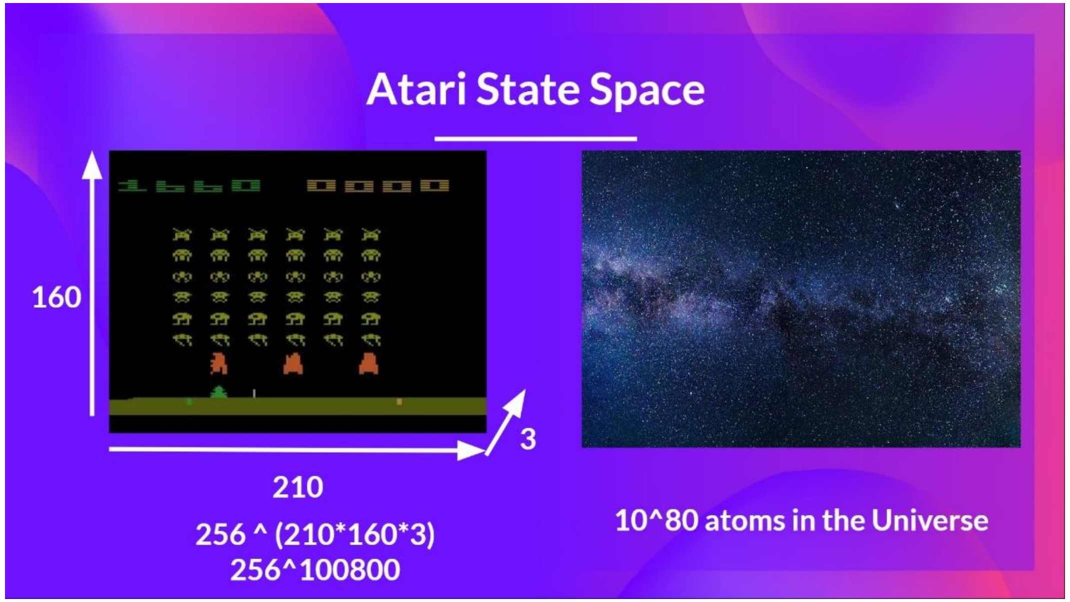
❖ Q-Learning was working well with small state space environments like:

✓ hence creating and updating a Q-table for that environment would not be efficient

- FrozenLake, we had 14 states.
- Taxi-v3, we had 500 states

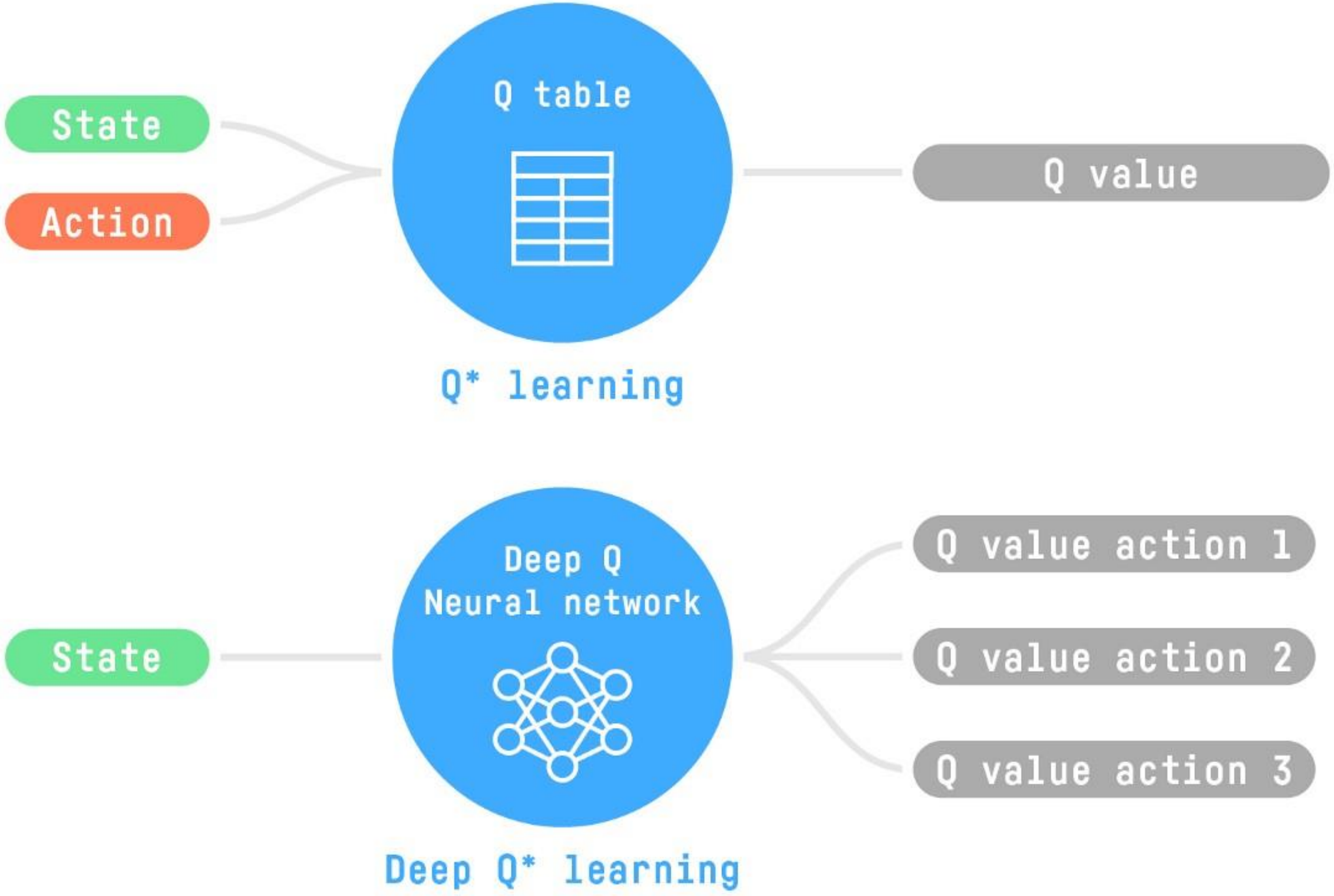
From Q-Learning to Deep Q-Learning





source: https://huggingface.co/blog/deep-rl-dqn

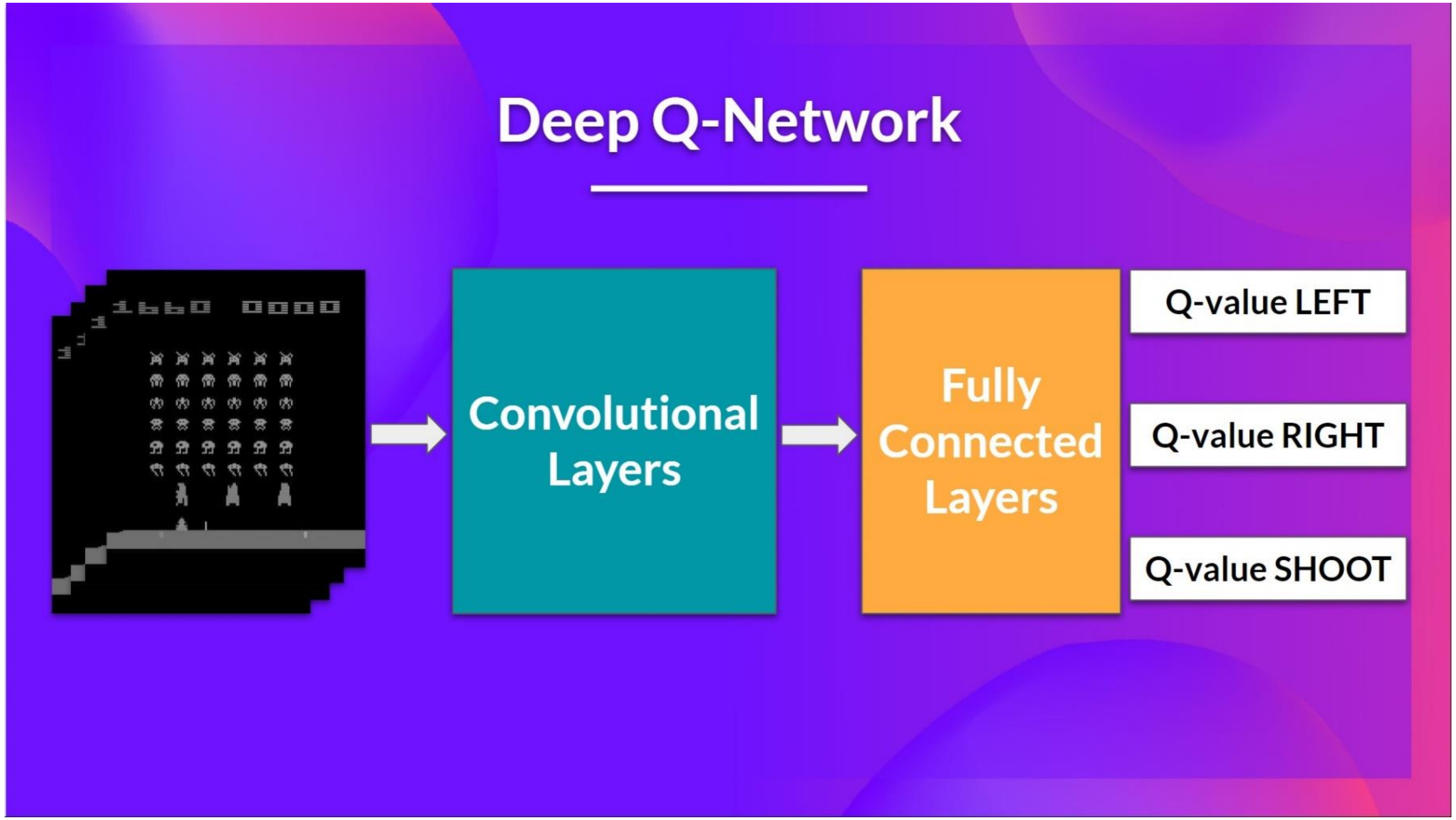
From Q-Learning to Deep Q-Learning



source: https://huggingface.co/blog/deep-rl-dqn

The Deep Q-Network (DQN)

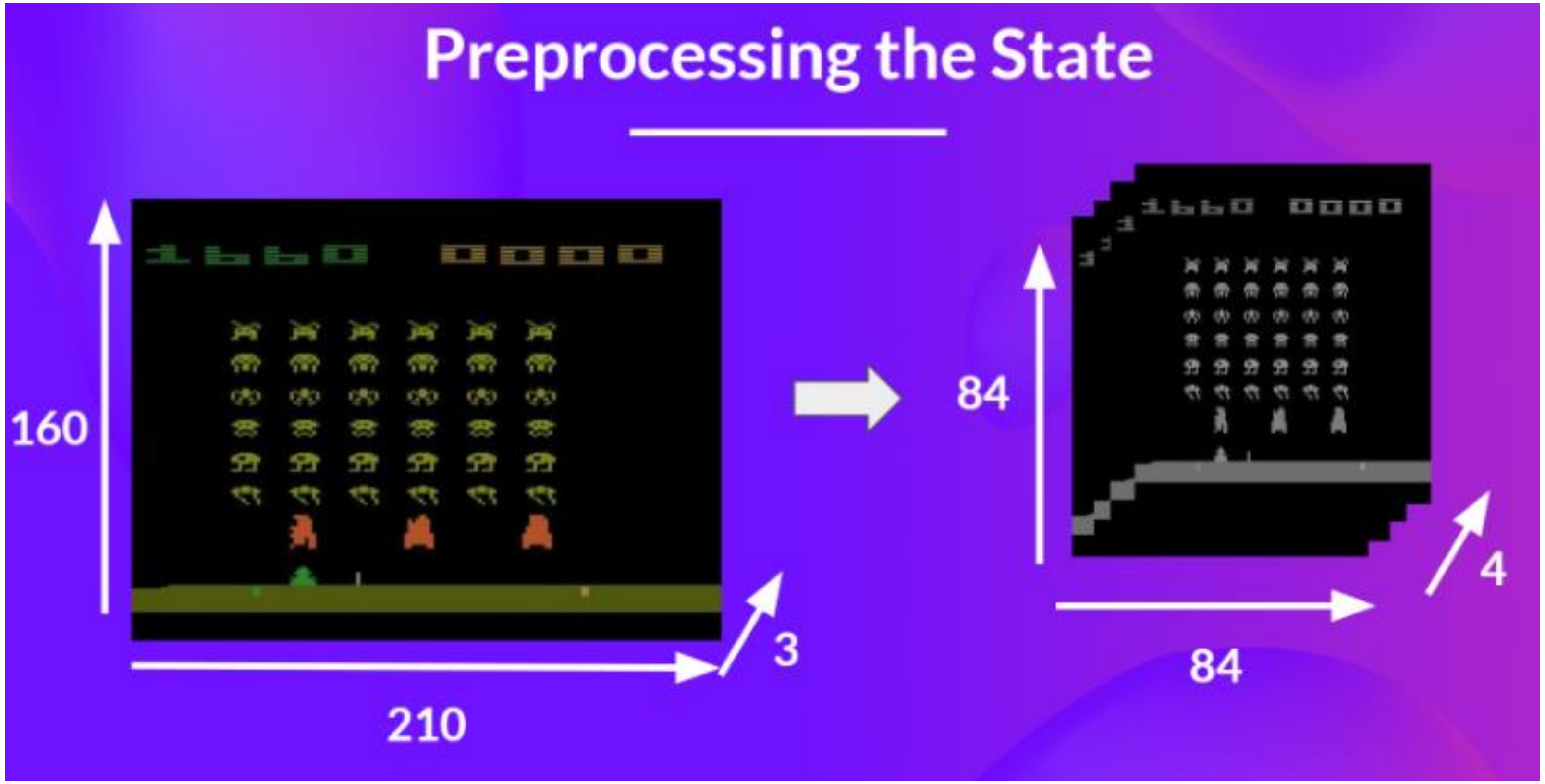
✓ As input, we take a **stack of 4 frames** passed through the network as a state and output a **vector of Q-values for each possible action at that state**.



source: https://huggingface.co/blog/deep-rl-dqn

❖ We mentioned that we preprocess the input

✓ what we do is **reduce the state space to 84x84 and grayscale it**

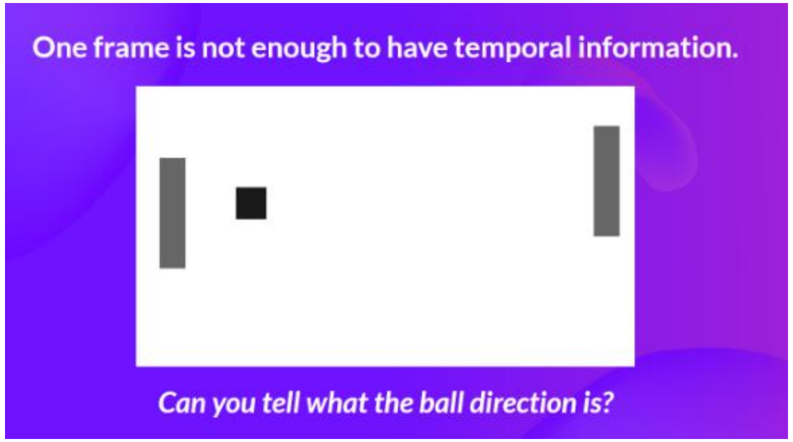


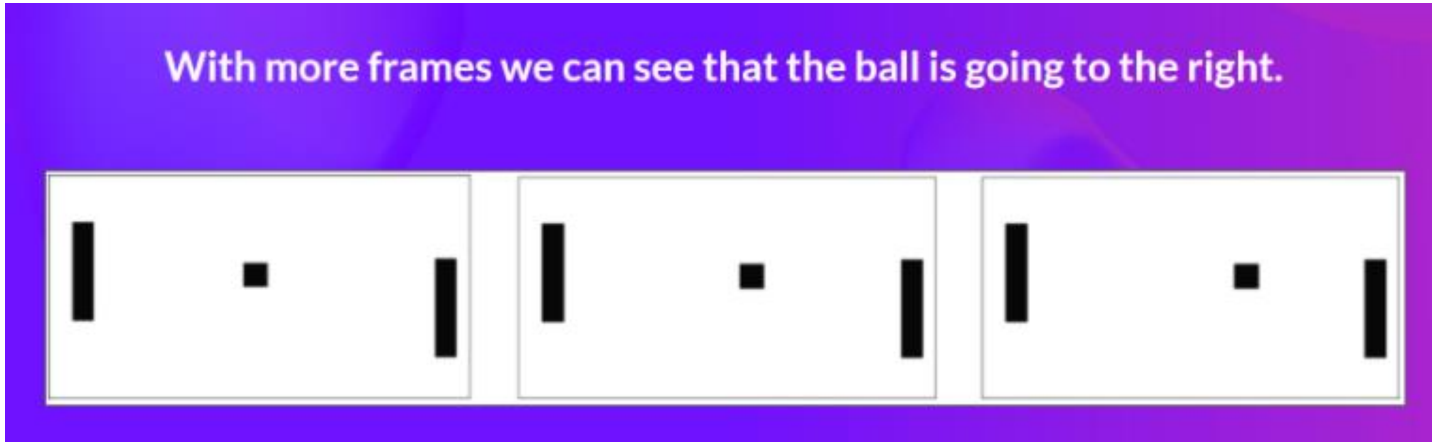
source: https://huggingface.co/blog/deep-rl-dqn

Why do we stack four frames together?

We stack frames together because it helps us **handle the problem of temporal limitation**

- Let's take an example with the game of Pong
- what if I add three more frames? Here you can see that the ball is going to the right.





source: https://huggingface.co/blog/deep-rl-dqn

The Deep Q-Network (DQN)

- **Deep Q-Learning is using a neural network to approximate, given a state, the different Q-values for each possible action at that state**
  - ✓ Three convolutional layers
    - These layers **allow us to capture and exploit spatial relationships in images**
    - But also, because frames are stacked together, **you can exploit some spatial properties across those frames.**
  - ✓ We have a couple of fully connected layers
    - That output a Q-value for each possible action at that state.

source: https://huggingface.co/blog/deep-rl-dqn

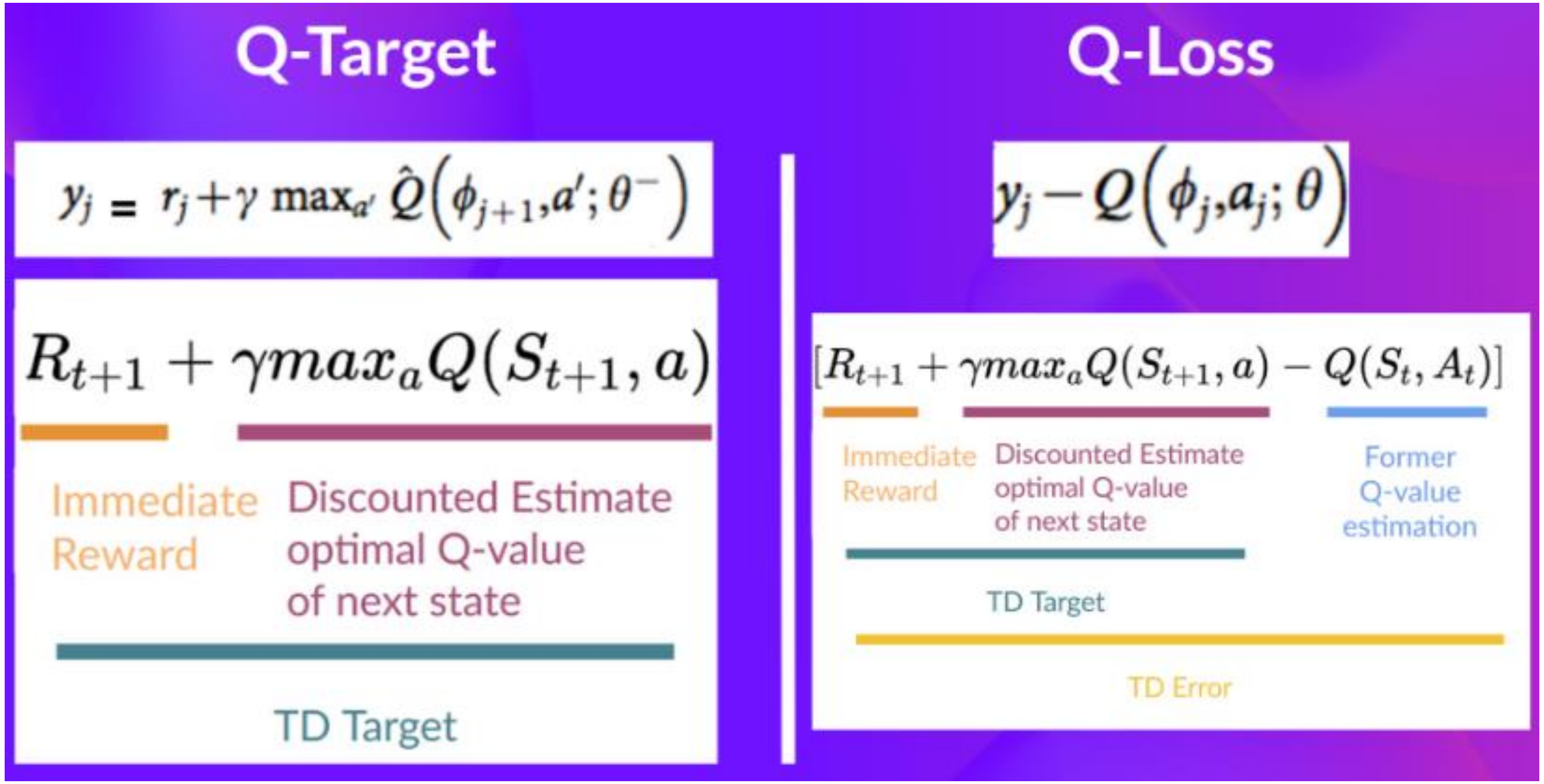
The Deep Q-Learning Algorithm

- Deep Q-Learning uses a deep neural network to approximate the different Q-values for each possible action at a state (value-function estimation).
- The difference is that, during the training phase, instead of updating the Q-value of a state-action pair directly as we have done with Q-Learning

$$\underbrace{Q(S_t, A_t)}_{\text{New Q-value estimation}} \leftarrow \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}} + \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{[\underbrace{\underbrace{R_{t+1}}_{\text{Immediate Reward}} + \underbrace{\gamma max_a Q(S_{t+1}, a)}_{\text{Discounted Estimate optimal Q-value of next state}}}_{\text{TD Target}} - \underbrace{Q(S_t, A_t)}_{\text{Former Q-value estimation}}]}_{\text{TD Error}}$$

source: https://huggingface.co/blog/deep-rl-dqn

- We create a Loss function between our Q-value prediction and the Q-target
  - ✓ Use Gradient Descent to update the weights of our DQN to approximate our Q-values better.



source: https://huggingface.co/blog/deep-rl-dqn

The Deep Q-Learning Algorithm

- Deep Q-Learning training algorithm has *two phases*:
  - ✓ **Sampling**: we perform actions and **store the observed experiences tuples in a replay memory.**
  - ✓ **Training**: Select the **small batch of tuple randomly and learn from it using a gradient descent update step.**
- Deep Q-Learning training might suffer from instability,
  - ✓ mainly because of combining a non-linear Q-value function (Neural Network)
  - ✓ and bootstrapping (when we update targets with existing estimates and not an actual complete return).

Three different solutions for DQN

❖ *Experience Replay:*
  ✓ to make more **efficient use of experiences.**

❖ *Fixed Q-Target* :
  ✓ **to stabilize the training.**

❖ *Double Deep Q-Learning:*
  ✓ to **handle the problem of the overestimation of Q-values.**

1) Experience Replay

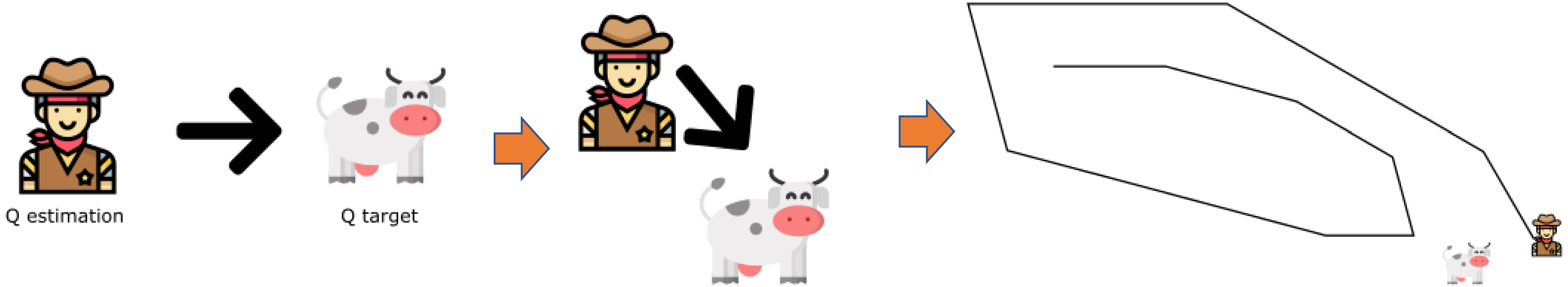
- Why do we create a replay memory?
- Experience Replay in Deep Q-Learning has two functions:
  - ✓ **With experience replay,** we create a replay buffer that saves experience samples **that we can reuse during the training.**
    - This allows us to **learn from individual experiences multiple times.**
  - ✓ **Avoid forgetting previous experiences and reduce the correlation between experiences.**
    - The problem we get if we give sequential samples of experiences to our neural network is that it tends to forget **the previous experiences as it overwrites new experiences.**

  - ✓ By randomly sampling the experiences, we remove correlation in the observation sequences and avoid **action values from oscillating or diverging catastrophically.**

KISTI SINCE 1962

When we calculate the loss between the Q-Target and the current Q-value

- ✓ Using the Bellman equation, we saw that the TD target is just the reward of taking that action at that state plus the discounted highest Q value for the next state.
- ✓ At every step of training, **our Q values shift but also the target value shifts**



Solution:

- ✓ Use a **separate network with a fixed parameter** for estimating the TD Target
- ✓ **Copy the parameters from our Deep Q-Network at every C step** to update the target network.

3) Double DQN

- This method handles the problem of the overestimation of Q-values.
- We face a simple problem by calculating the TD target:
  - ✓ how are we sure that **the best action for the next state is the action with the highest Q-value?**
- The solution is: when we compute the Q target, we use two networks to decouple the action selection from the target Q value generation
  - ✓ Use our **DQN network** to select the best action to take for the next state (the action with the highest Q value).
  - ✓ Use our **Target network** to calculate the target Q value of taking that action at the next state.