Introduction to Deep Reinforcement Learning

IA Frameworks

Brendan Guillouet - 07/12/2020

Institut National des Sciences Appliquées de Toulouse

Tools

ML Python Libraries

scikit learn

Keras

gensim

surprise

Gym

TensorFlow

NLTK

Python Environment

PC

jupyter

ANACONDA

Viz' Python Libraries

matplotlib
Version 3.1.1

seaborn

plotly

Framework & Tool

docker

Google Cloud Platform

git

Table of contents

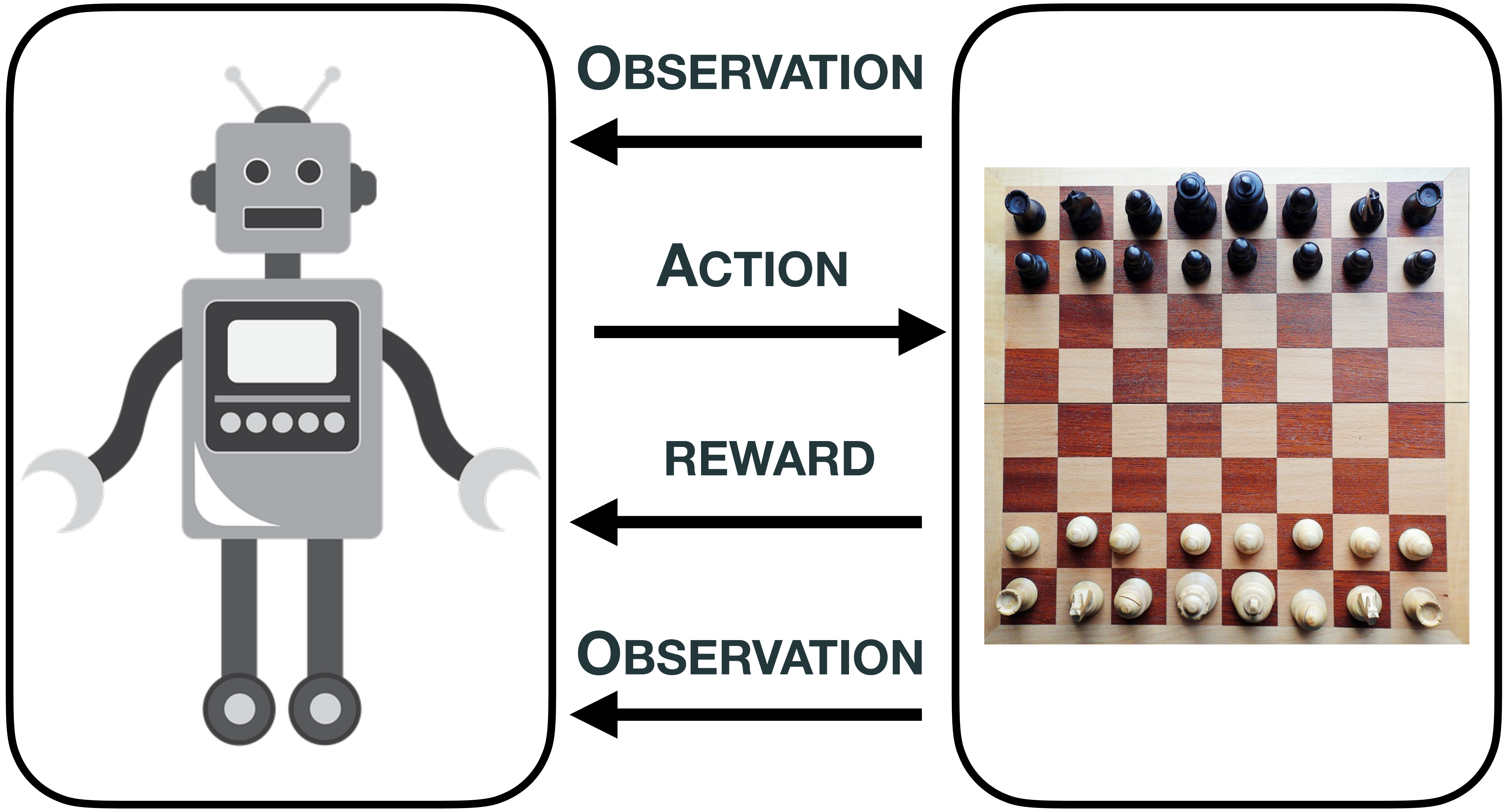
Introduction

Definitions

In reinforcement learning, an agent makes **OBSERVATIONS** of the **STATES** of an **ENVIRONMENT**.

It takes **ACTION** within the **ENVIRONMENT** and receives **REWARDS**.

Example: Chess Game



OBSERVATION: Locations of each pieces on the chessboard (32 x (id, x, y)).

ACTION: Move piece p from (x_a, y_a) to (x_b, y_b).

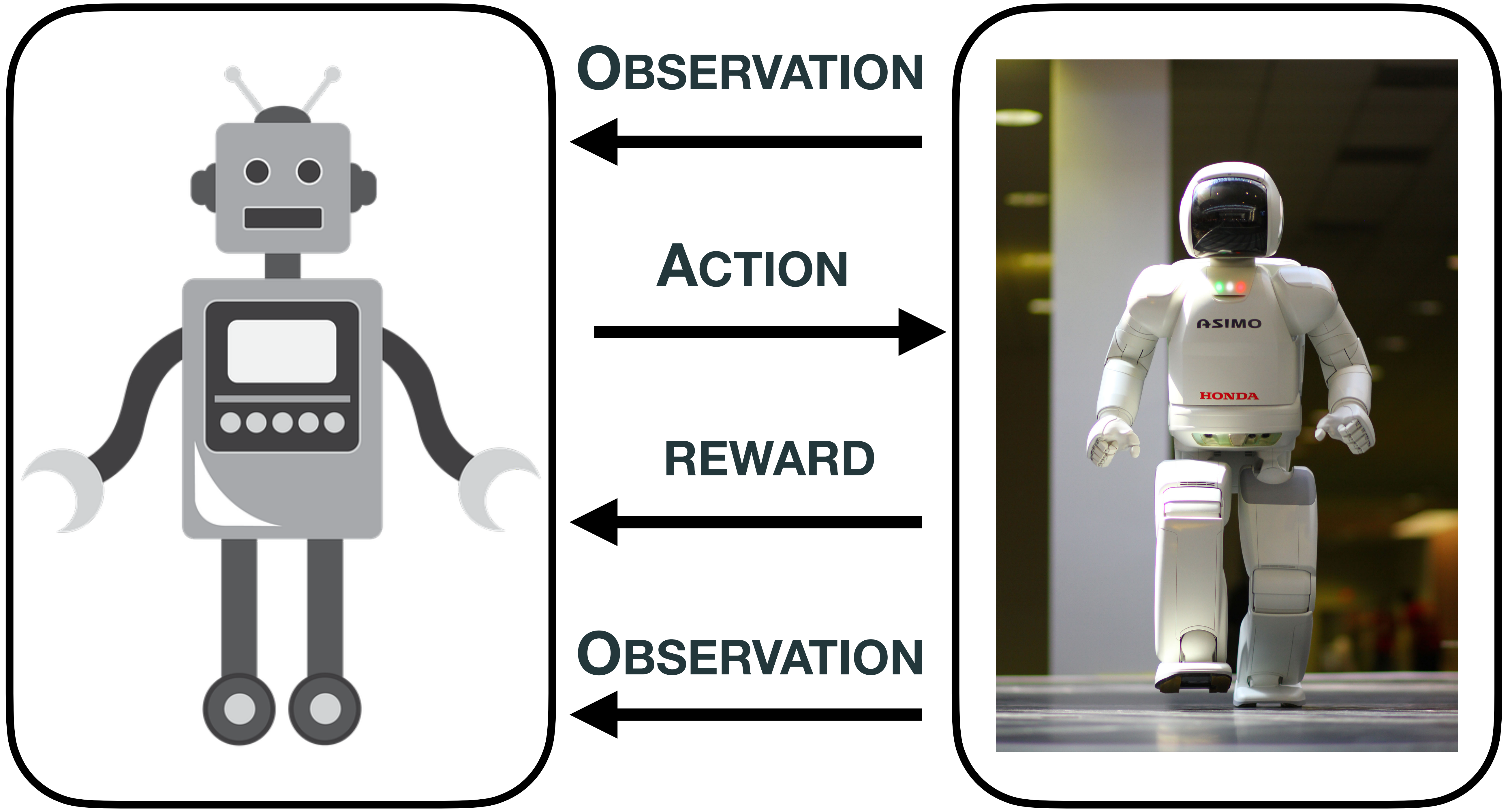
REWARD: Positive if a piece has been token.

Definitions

In reinforcement learning, an agent makes **OBSERVATIONS** of the **STATES** of an **ENVIRONMENT**.

It takes **ACTION** within the **ENVIRONMENT** and receives **REWARDS**.

Example: Walking Robot



Observation: Measurement from various sensor (time series).

Action: Move members (3D).

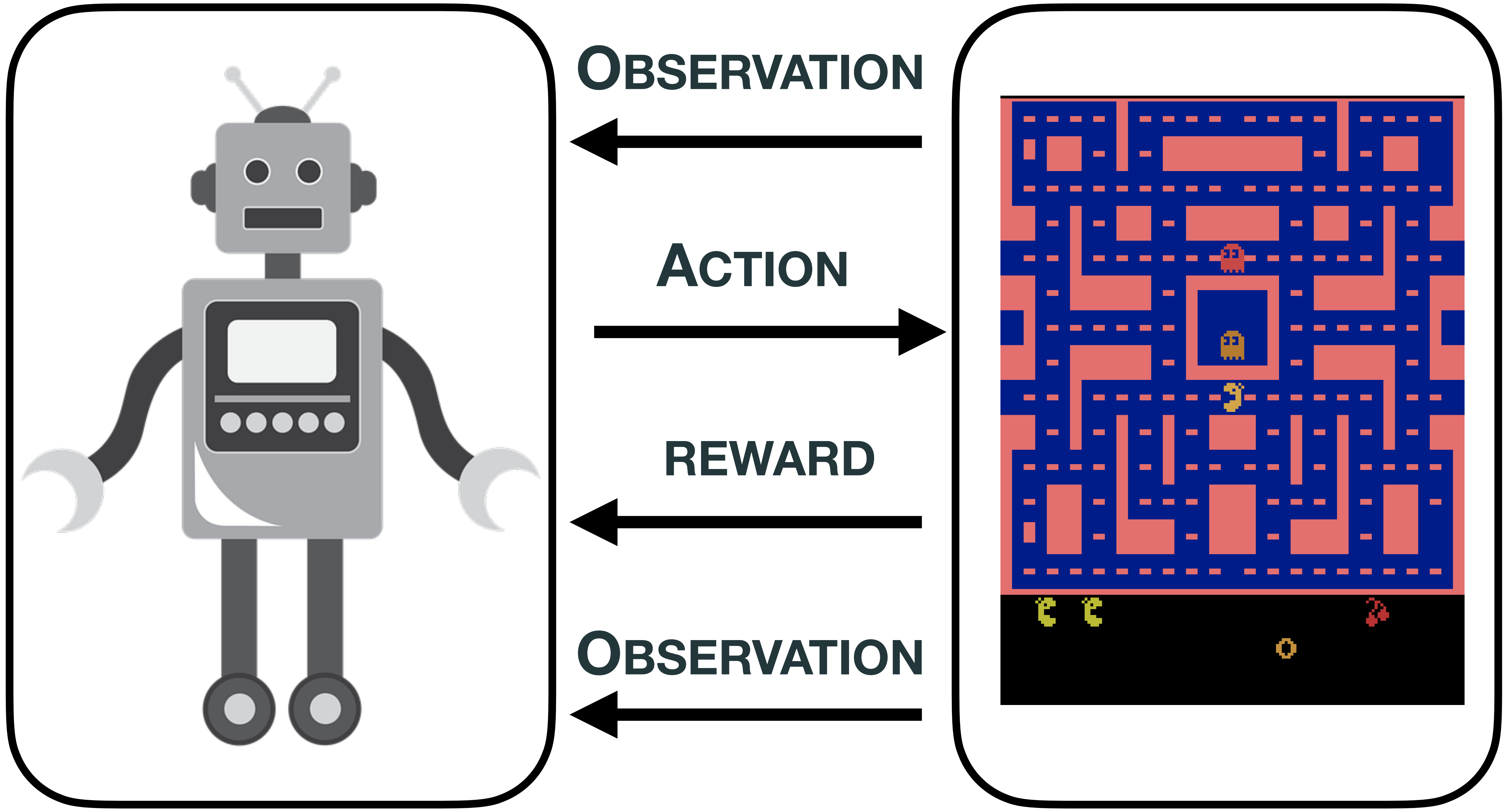
Reward: Positive when it approach the destination. Negative otherwise.

Definitions

In reinforcement learning, an agent makes **OBSERVATIONS** of the **STATES** of an **ENVIRONMENT**.

It takes **ACTION** within the **ENVIRONMENT** and receives **REWARDS**.

Example: Pac Man



OBSERVATION: The image itself.

ACTION: left/right/up/down/nothing.

REWARD: Game Point.

Definitions

When using reinforcement learning you need to define:

An **Agent**. The one who will take action within the environment.

An **Environment** where the action will be taken.

> The **State** of the environment defines it after each action.
>
> The **Observation** is what we will used from the state to decide the next action.

A list of possible **Actions** actions that will affect the environment and produce a new state.

A **Reward**. A numerical value which reveal how positive or negative the action is.

You can’t apply reinforcement learning if you’re not able to define these objects properly!

Objective

Maximise the long-term rewards.

Do not pull all the effort on capturing the queen if it means losing all you pieces.

How ? There are two mains approaches**:**

Value-Based. Look for the optimal **reward**.

- *Learn to estimate the expected rewards for each action in each state.*
- *Use this knowledge to choose the best action.*

Policy-Based. Look for the optimal **policy**.

- *Learn directly the best action to take for each observation.*

NB: Methods like Actor-Critic try to optimise both policy and rewards.

Objective

Maximise the long-term rewards.

Do not pull all the effort on capturing the queen if it means losing all you pieces.

How ? There are two mains approaches**:**

Value-Based. Look for the optimal reward.

- ***Learn to estimate the expected rewards for each action in each state.***
- ***Use this knowledge to choose the best action.***

Policy-Based. Look for the optimal **policy**.

- *Learn directly the best action to take for each observation.*

NB: Methods like Actor-Critic try to optimise both policy and rewards.

Definitions

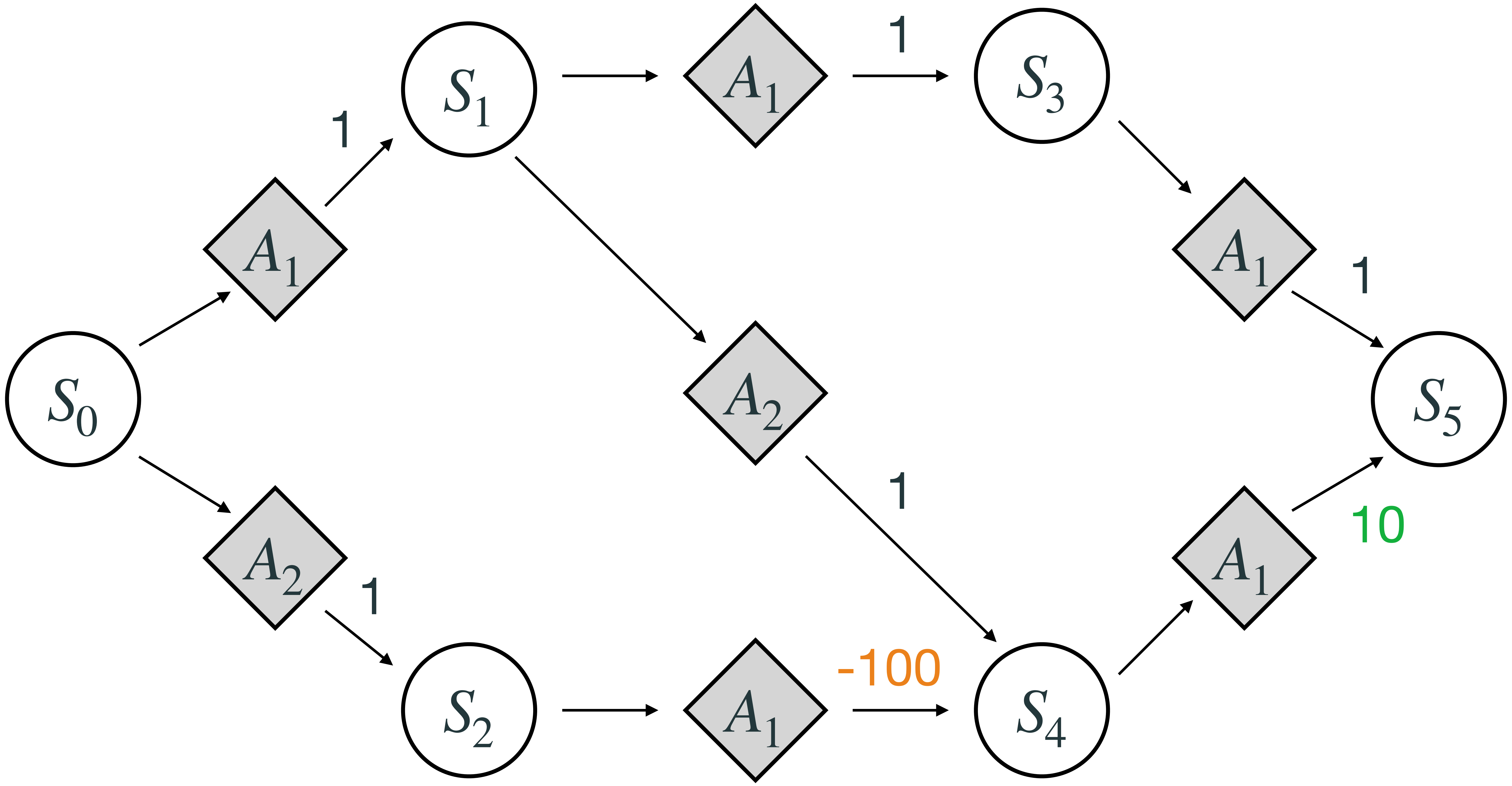
Markov Decision Process - Definition

A **M**arkov **D**ecision **P**rocess is composed of :

- A set of state $S = \{s_1, s_2, ..., s_n\}$.
- A set of action $A = \{a_1, a_2, ..., a_m\}$
- A reward function : $R = S \times A \times S \rightarrow \mathscr{R}$
- A transition function: $P_{ij}^{a} = P(s_j \,|\, s_i, a_k)$

Markov Decision Process - Example

- A set of state $S = \{1,2,3,4,5\}$.
- A set of action $A = \{1,2\}$
- A reward function : $R = \left[\{0,A,1\} = 1,\{0,B,2\} = 2,\ldots\right]$
- A transition function: $P_{ij}^{a} = 1, \forall\ (i, j, a),\ 0$ otherwhise



Policies

There are 3 policies for this **MDP**.

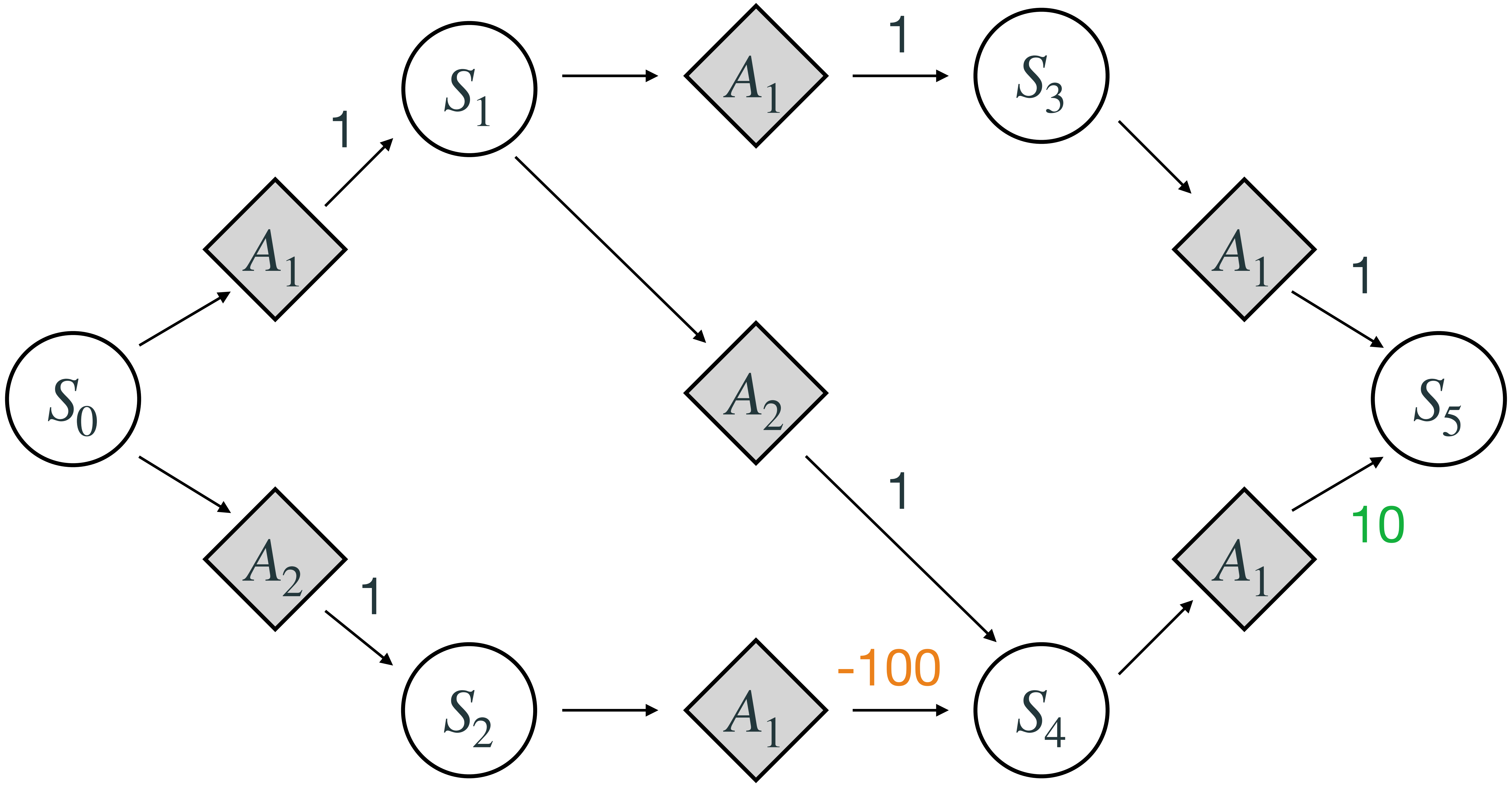
$\Pi_1 = \{0 \rightarrow 1 \rightarrow 3 \rightarrow 5\}$ $\quad R = 3$

$\Pi_2 = \{0 \rightarrow 1 \rightarrow 4 \rightarrow 5\}$ $\quad R = 12$

$\Pi_3 = \{0 \rightarrow 2 \rightarrow 4 \rightarrow 5\}$ $\quad R = -89$

Which one is the best?

Compute their total reward !



Policies

There are 3 policies for this **MDP**.

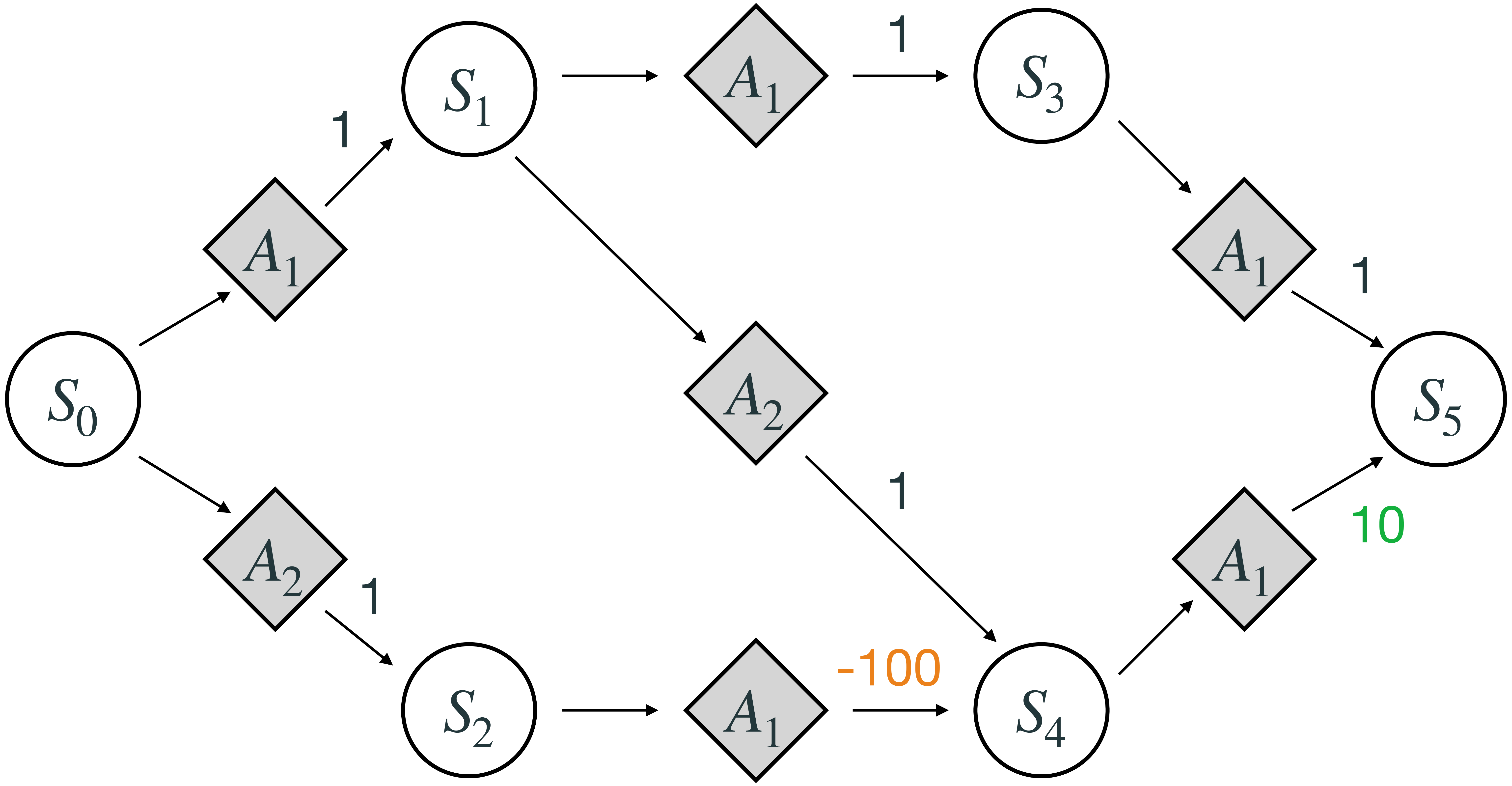
$\Pi_1 = \{0 \rightarrow 1 \rightarrow 3 \rightarrow 5\} \quad R = 3$

$\Pi_2 = \{0 \rightarrow 1 \rightarrow 4 \rightarrow 5\} \quad R = 12$

$\Pi_3 = \{0 \rightarrow 2 \rightarrow 4 \rightarrow 5\} \quad R = -89$

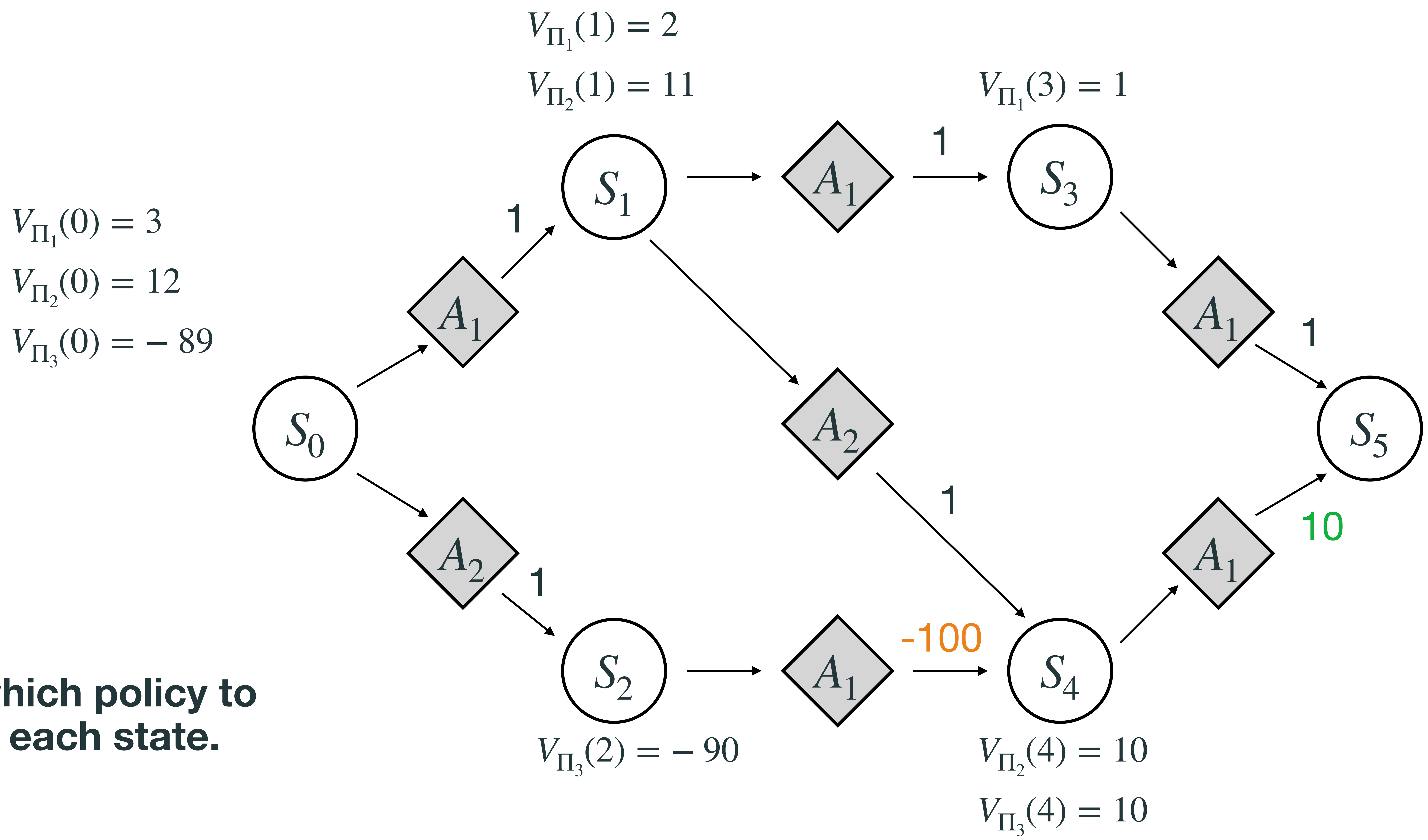
Which one is the best?

Compute their total reward !



SATE VALUE

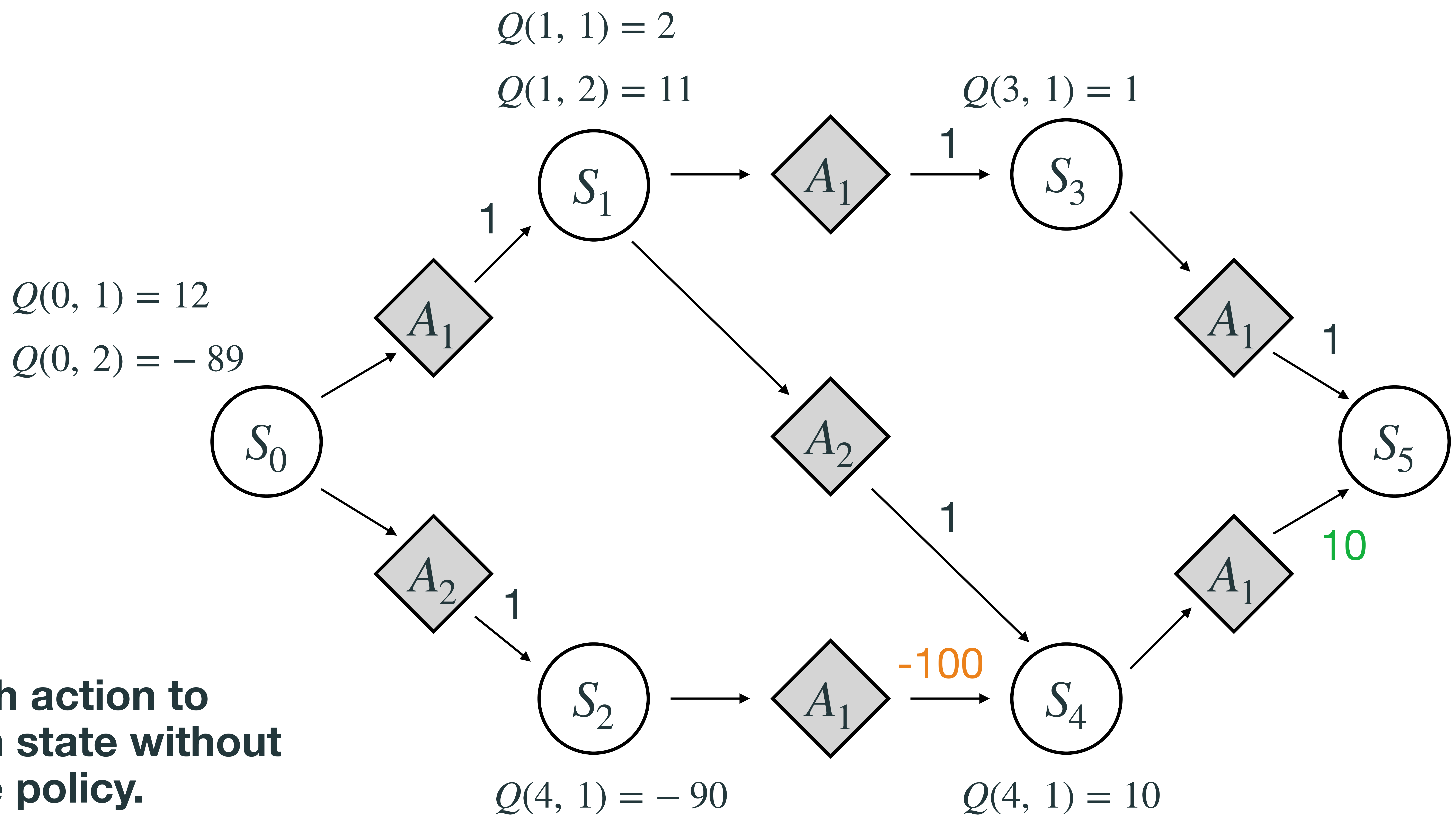
A state value $V_\Pi(s)$ describes how good is it to run policy Π from state s.



Define which policy to follow at each state.

Action-State value

An action-state value $Q(s, a)$ describes the value of taking and action a from state s.



Define which action to take at each state without knowing the policy.

general Definition

State value of a policy π:

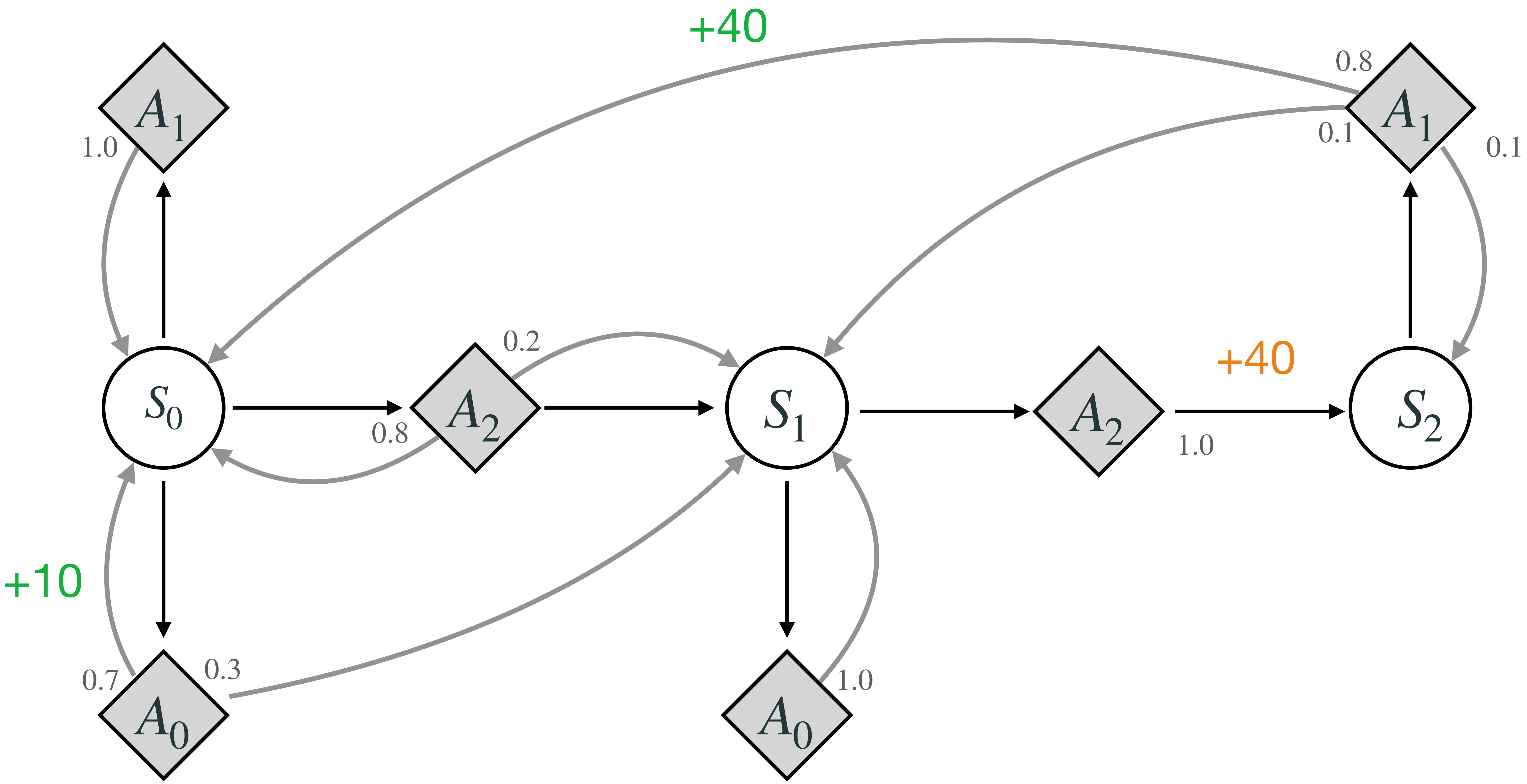
$$V^{\pi}(s) = \mathbb{E}\left(\lim_{H \to \infty} \sum_{t=0}^{H} \gamma^t r_t \,\middle|\, s_0 = s, \pi \right)$$

Action-state value function of a policy

$$Q^{\pi}(s, a) = \mathbb{E}\left(\lim_{H \to \infty} \sum_{t=0}^{H} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a, \pi \right)$$

Action-State value - Probabilistic Action

What if your action can lead you to different state ?



From state $= S_0$

$P^{A_0}_{S_0,S_0} = 0.7 \quad P^{A_0}_{S_0,S_1} = 0.3$

$P^{A_1}_{S_0,S_0} = 1$

$P^{A_2}_{S_0,S_0} = 0.8 \quad P^{A_2}_{S_0,S_1} = 0.2$

From state $= S_1$

$P^{A_0}_{S_1,S_1} = 1$

$P^{A_2}_{S_0,S_2} = 1$

From state $= S_2$

$P^{A_1}_{S_2,S_0} = 0.8 \quad P^{A_1}_{S_2,S_1} = 0.1 \quad P^{A_1}_{S_2,S_2} = 0.1$

SOLUTION: The Bellman Equation

Bellman Optimality Equation

For all s :

$$V^{\star}(s) = max_a \sum_{s'} P^a_{s,s'} \big[R(s,a,s') + \gamma \cdot V^{\star}(s') \big]$$

- $V^{\star}$ is the optimal state value at state s.
- $P^a_{s,s'}$ is the transition probability from state s to state s' given that the agent choice action a.
- $R(s,a,s')$ is the reward that the agent gets when it goes from state s to state s' given that the agent choose action a.
- γ is the discount rate (t*he importance we give to future rewards).*

Bellman Optimality Equation

For all s :

$$V^{\star}(s) = max_a \sum_{s'} P^{a}_{s,s'} \left[\boldsymbol{R(s, a, s') + \gamma \cdot V^{\star}(s')} \right]$$

- $V^{\star}$ is the optimal state value at state s.
- $P^{a}_{s,s'}$ is the transition probability from state s to state s' given that the agent choice action a.
- $R(s, a, s')$ is the reward that the agent gets when it goes from state s to state s' given that the agent choose action a.
- γ is the discount rate (t*he importance we give to future rewards).*

Bellman Optimality Equation

For all s :

$$V^{\star}(s) = max_a \sum_{s'} \boldsymbol{P^{a}_{s,s'}} \big[\boldsymbol{R(s,a,s') + \gamma \cdot V^{\star}(s')} \big]$$

- $V^{\star}$ is the optimal state value at state s.
- $P^{a}_{s,s'}$ is the transition probability from state s to state s' given that the agent choice action a.
- $R(s,a,s')$ is the reward that the agent gets when it goes from state s to state s' given that the agent choose action a.
- γ is the discount rate (t*he importance we give to future rewards).*

Bellman Optimality Equation

For all s :

$$V^{\star}(s) = max_a \sum_{s'} P^a_{s,s'}\left[R(s,a,s') + \gamma \cdot V^{\star}(s')\right]$$

- $V^{\star}$ is the optimal state value at state s.
- $P^a_{s,s'}$ is the transition probability from state s to state s' given that the agent choice action a.
- $R(s,a,s')$ is the reward that the agent gets when it goes from state s to state s' given that the agent choose action a.
- γ is the discount rate (t*he importance we give to future rewards).*

How to estimate this optimal sates? The **Value Iteration algorithm.**

Iteration algorithm

Iteration Algorithm

Value Iteration algorithm.

- Initialise: $V_0(s) = 0, \forall s$.
- Iterate, $\forall s$:

$$V_{k+1}(s) \leftarrow max_a \sum_{s'} P^a_{s,s'} \left[R(s,a,s') + \gamma \cdot V_k(s') \right]$$

- $V_k(s)$ is the estimated value of state s at the k^{th} iteration of the algorithm.
- Algorithm is guaranteed to converge to the optimal state value (given enough time)

We can now evaluate the optimal policy…

… but we don't know which action to take at each state !

Iteration Algorithm

Q-Value Iteration algorithm.

- Initialise: $Q_0(s,a) = 0,\ \forall(s,a)$.
- Iterate, $\forall(s,a)$:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P^{a}_{s,s'}\left[R(s,a,s') + \gamma \cdot max_{a'}\ Q_k(s',a')\right]$$

Once you have the optimal ***Q-value***, $Q*$, the ***optimal policy***,$\Pi*(s)$, is defined as:

$$\Pi*(s) = argmax_a\ Q*(s,a)$$

Real application

Problem: The **Q-value iteration algorithm** required to know the complete MDP:

- complete list of possible states $s \in S$ and actions $a \in A$,
- all transition probabilities $T(s, a, s')$
- all possible reward $R(s, a, s')$

In real case, the agent has only partial knowledge of the MDP:

- complete list of possible states $s \in S$ and actions $a \in A$,
- $T(s, a, s')$ and $R(s, a, s')$ are unknown

Proposition: Estimate those values

- $R(s, a, s')$ requires to see each transition at least once.
- $T(s, a, s')$ requires to experience it multiple times.

Q-Learning

Temporal Difference Learning

TD-Learning iteration algorithm

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha V^*_{k+1}(s)$$

- Initialise: $V_0(s) = 0, \ \forall s$.
- At iteration k, we have an estimation of the optimal state value $V_k(s)$, then $\forall s$:
  - Choose a reachable state s' *(we'll come back to that later)*.
  - Compute an estimation of the next state value $V^*_{k+1}(s) = R(s, a, s') + \gamma \cdot V_k(s')$
  - Update V_{k+1} as a combination of the previous estimation $V_k(s)$ and the estimation $V^*_{k+1}(s)$

  Where the learning rate $\alpha \in [0,1]$

Temporal Difference Learning VS Value Iteration Algorithm

TD-Learning iteration algorithm.

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha\left[R(s,a,s') + \gamma \cdot V_k(s')\right]$$

$$V_{k+1}(s) \leftarrow (1-\alpha)V_k(s) + \alpha V^*_{k+1}(s)$$

Value Iteration algorithm.

$$V_{k+1}(s) \leftarrow max_a \sum_{s'} P^a_{s,s'}\left[R(s,a,s') + \gamma \cdot V_k(s')\right]$$

$$V_{k+1}(s) \leftarrow max_a \sum_{s'} P^a_{s,s'} V^*_{k+1}(s)$$

- The **discount rate** γ represent how much you can trust the features reward.
- The **learning rate** α represent how much you can trust the next state value estimation.

The Q-value iteration algorithm can be adapted the same way.

Q-Learning

Q-Learning iteration algorithm

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left[R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right]$$

- We build a memory table to store **Q-value** for all possible combinations of s and a.
- As we keep playing we update this table.
- And it will converge…
- …with some tricks!

Off Policy

Off-Policy algorithm

The **TD-Learning** and **Q-learning** iteration algorithm used with random policy exploration.

Π_l : The policy you are learning *(by updating either the Q-value or the state value).*

Π_e : The policy you are executing *(to choose the action you're taking).*

- Π_e is random when using off-policy algorithm
- You are learning how to act by observing someone doing random action.
- Q-learning will learn only if the exploration policy explores the MDP enough.
- It may take and extremely long time to do so..

Can't we do better?

Exploration policy : $\epsilon - greedy$ policy

At each step you will choose either:

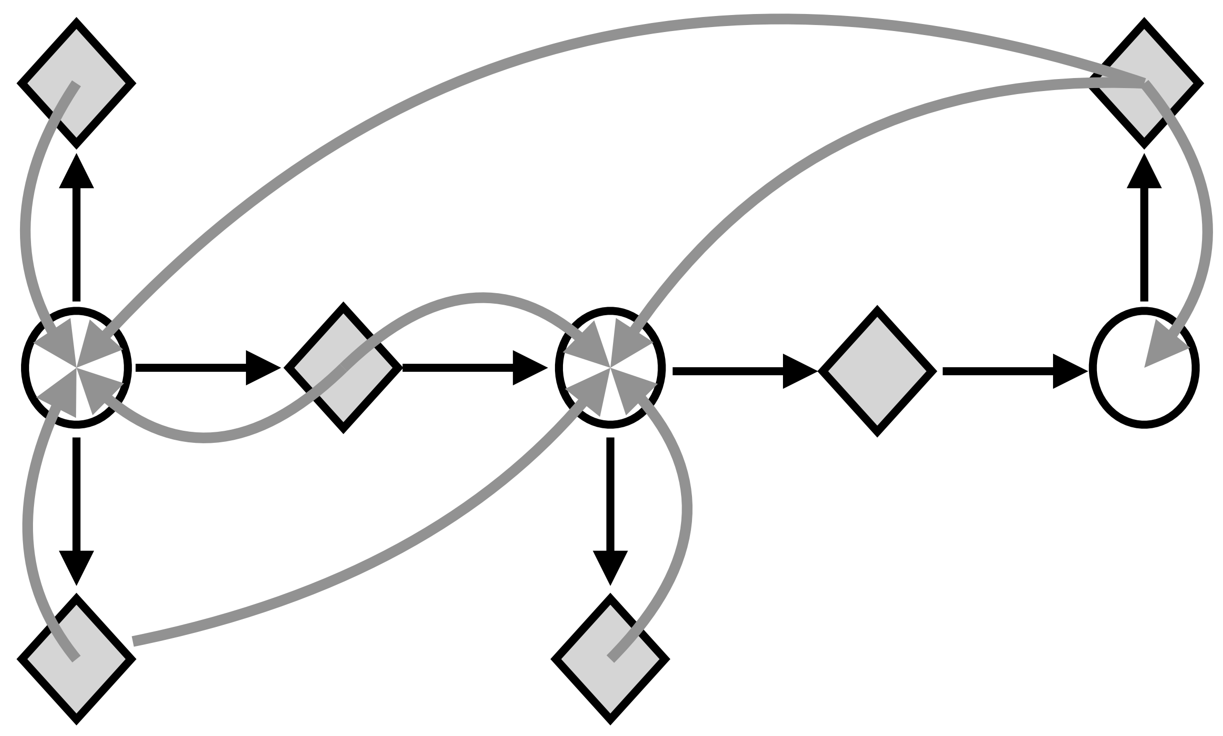
- The exploration policy (random one) Π_e with a probability ϵ
- The learned policy Π_l with a probability $(1 - \epsilon)$

You start with $\epsilon = 1$: purely random exploration of the environment.

You decrease the value of epsilon over iteration to explore less and exploit more the interesting pars of the environment

TP

- *Q_learning.ipynb :* Implement **Q-VALUE ITERATION ALGORITHM** and ***Q-LEARNING ITERATION ALGORITHM*** on the toy MDP of this presentation



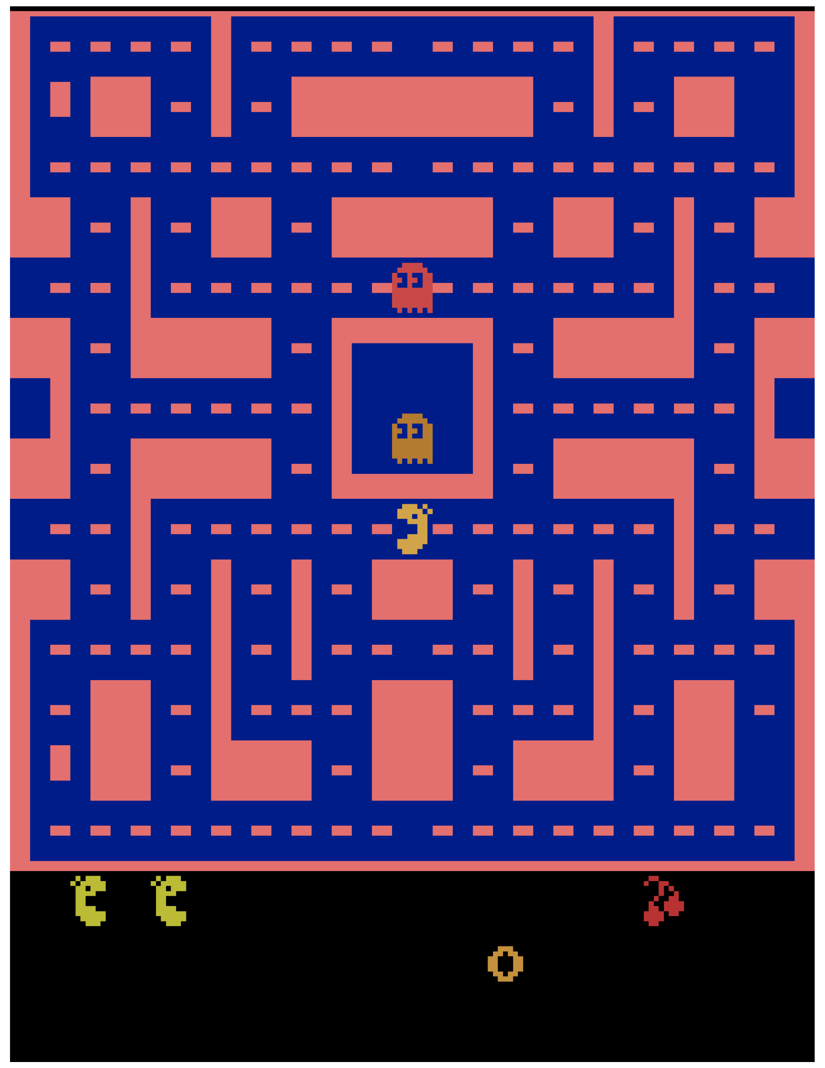
Deep Q-Learning (DQN)

Approximate Q-Learning

Problem: Q-learning still does not scale to large MDPs

- $T(s, a, s')$ and $R(s, a, s')$ are not computed for every (s, a) couple.
- But $Q(s, a)$ are!

Example: Pacman



- 240 pellets that can be eaten or here or not.
- It represent $2^{240} \approx 10^{72}$ configuration. *Number of atoms in the universe.*
- And we need to consider ghost and Pacman position...

We can' compute nor store Q-value for every (s, a) couple.

Approximate and Deep Q-learning

Objective: Use a function to evaluate the Q-value

Q-Learning

Store and update all Q-values in a big table

Pick value from it.

Approximate Q-Learning

We train a function Q_θ that will generalise the approximation of the Q-values table

Use this function to estimate value

Compute hand-crafted features (localisation of paceman, distance to the ghost, localisation of remaining pellets…)

Deep Q-Learning

Use a CNN network as a Q_θ function and use image as features.

DeepMind makes it work (very well) on various Atari Game in 2014.

Approximate and Deep Q-learning

Q-Value Iteration algorithm.

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P^a_{s,s'}\left[R(s,a,s') + \gamma \cdot max_{a'}\, Q_k(s',a')\right]$$

Q-Learning

$$\begin{aligned} target &= R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a') \\ Q_{k+1}(s,a) &\leftarrow (1-\alpha)Q_k(s,a) + \alpha\left[target\right] \end{aligned}$$

Approximate/Deep Q-Learning

$$target = R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')$$

At step k, generate $batch\_size\ \ targets$ and train model on this batch

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s\sim' P(s'|s,a)}[(Q_\theta(s,a) - target(s'))^2]_{\theta=\theta_k}$$

- θ_k are the weights of the (Deep) Q function computed at step k (batch)
- α is the learning rate and can be interpreted the same way.

Pseudo Code

- Create Q network.
- Iterate while $max\_number\_episode$ or $goal\_reward$ is not achieved:
  - Play one episode - Explore randomly or Exploit with Q according to $random\_probability$.
  - If we train more than $min\_pre\_trained\_episode$
    - Create target using Q.
    - Train the Q network.
    - Start decreasing $random\_probability$.

DEFINITIONS

EPISODE: One complete run of a game.

Pacman: all the move until the pacman died.

STEP: One move of the game.

Pacman: one action taken in the environment (up, down, left, right)

EXPERIENCE: All the information that resume one step: .

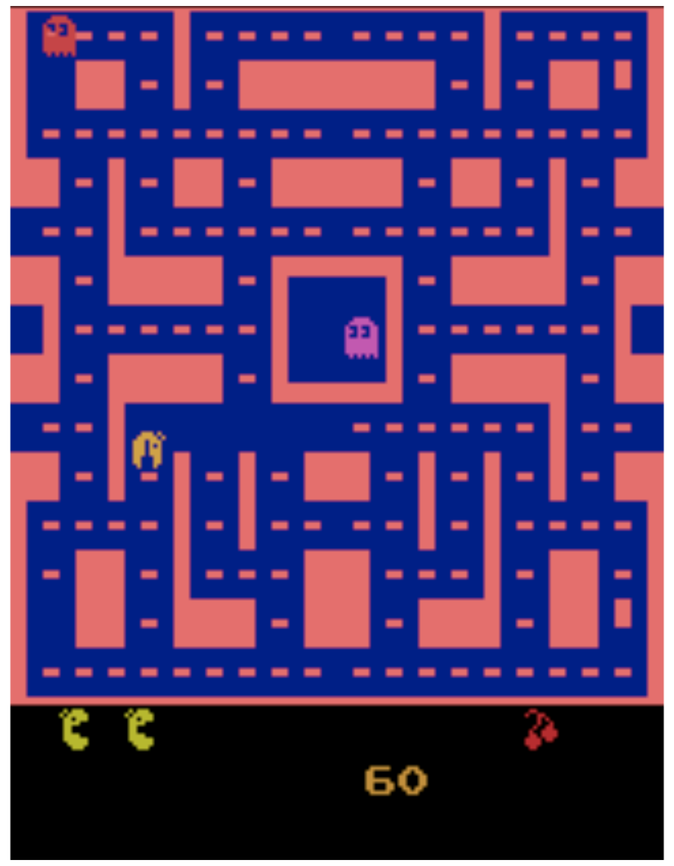
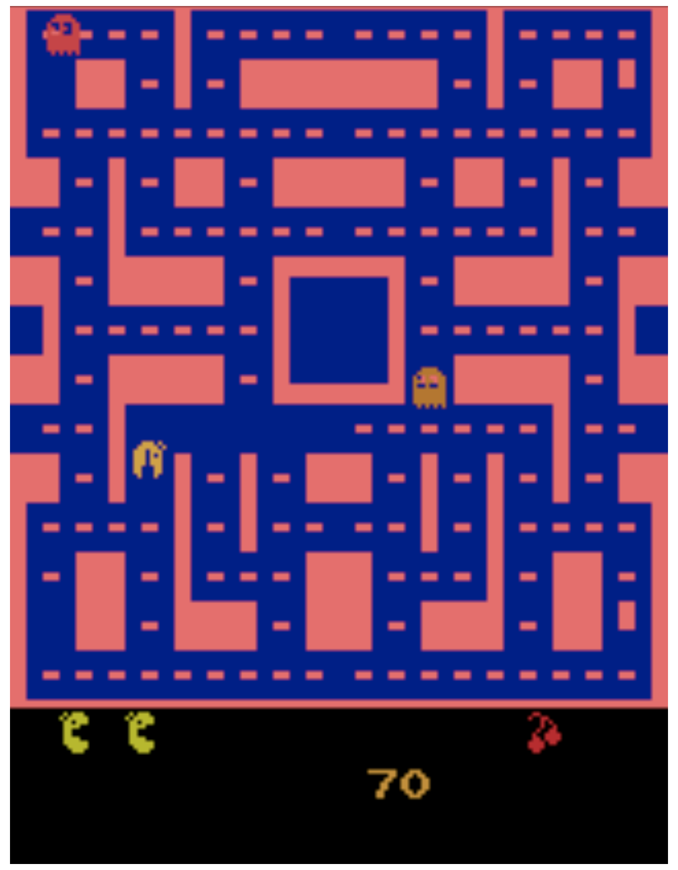
s: from state

a: action chosen

s': to state

r: reward obtained

d: is game over

Pacman	*From state*	*Action*	*To state*	*Reward*
		DOWN		10

Experiences for training dataset

$$\begin{array}{cccc} \textit{from state} & \textit{action} & \textit{to state} & \textit{reward} \end{array}$$

$$\begin{bmatrix} s_{0,0} & a_{0,0} & s'_{0,0} & r_{0,0} \\ s_{0,1} & a_{0,1} & s'_{0,1} & r_{0,1} \\ \dots & \dots & \dots & \dots \\ s_{0,ns_0} & a_{0,ns_0} & s'_{0,ns_0} & r_{0,ns_0} \\ s_{1,0} & a_{1,0} & s'_{0,1,0} & r_{1,0} \\ \dots & \dots & \dots & \dots \\ s_{ne,0} & a_{ne,0} & s'_{ne,0} & r_{ne,0} \\ \dots & \dots & \dots & \dots \\ s_{ne,ns_{ne}} & a_{ne,ns_{ne}} & s'_{ne,ns_{ne}} & r_{ne,ns_{ne}} \end{bmatrix}$$

- $\{s, a, s', r\}_{i,j}$, *from state, action, to state* and *reward* of episode i at step j.
- ne number of episodes in the training dataset.
- ns_i number of steps in episode i.
- Number of experience in the training dataset : $\sum_{i=0}^{ne} ns_i$

Problem: The input of the supervised learning problem are not $i.i.d.$

Experience Replay

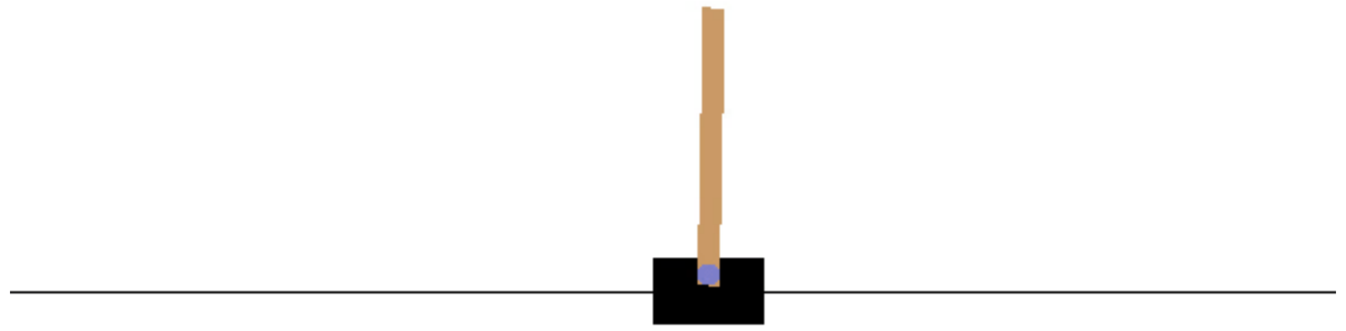
Solution: The Experience Replay

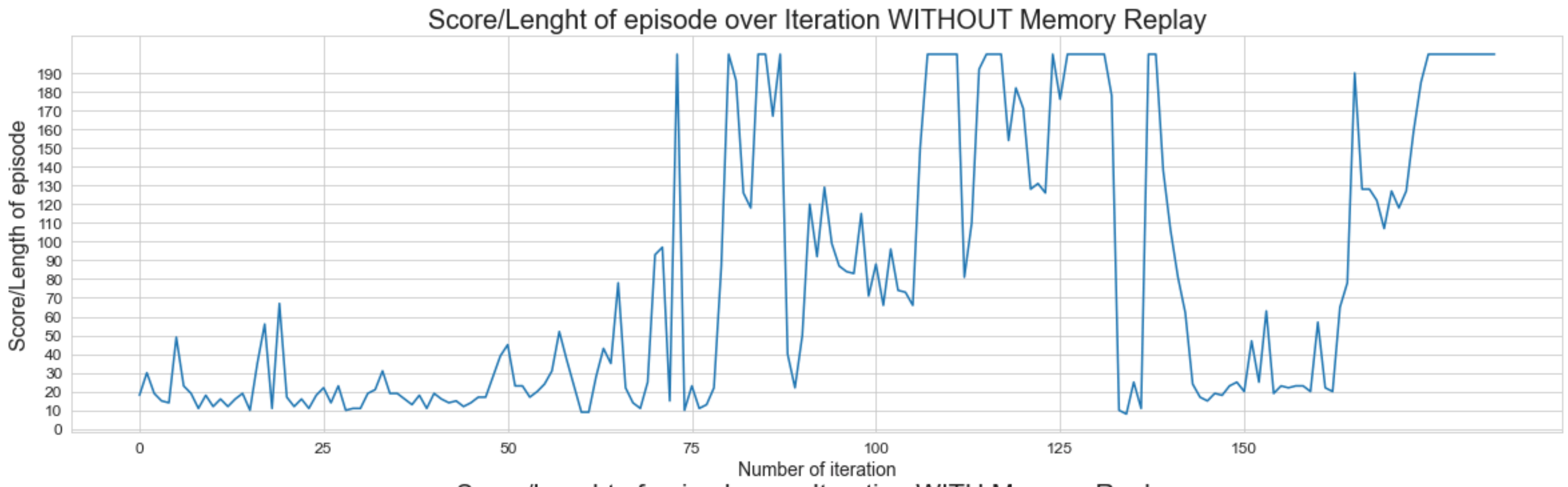
$$
\begin{array}{cccc}
\textit{from state} & \textit{action} & \textit{to state} & \textit{reward} \\
\end{array}
$$

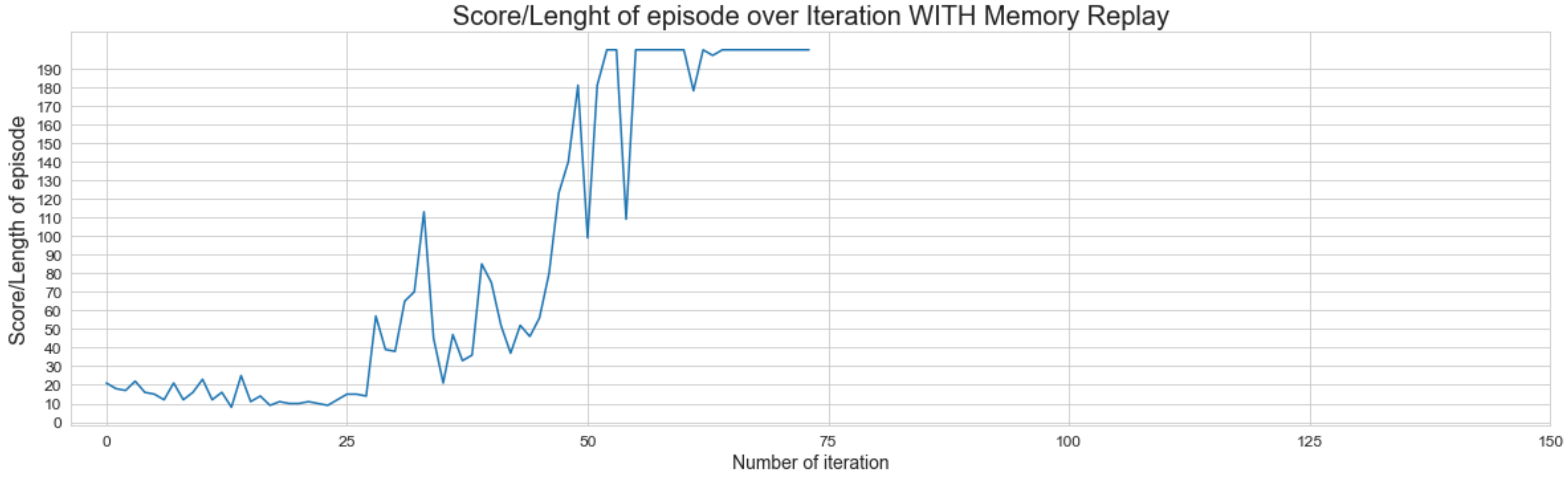
$$
\begin{bmatrix}
s_{10,3} & a_{10,3} & s'_{10,3} & r_{10,3} \\
s_{8,37} & a_{8,37} & s'_{8,37} & r_{8,37} \\
\dots & \dots & \dots & \dots \\
s_{3,17} & a_{3,17} & s'_{3,17} & r_{3,17} \\
s_{5,38} & a_{5,38} & s'_{5,38} & r_{5,38} \\
\dots & \dots & \dots & \dots \\
s_{8,6} & a_{8,6} & s'_{8,6} & r_{8,6} \\
\dots & \dots & \dots & \dots \\
s_{3,10} & a_{3,10} & s'_{3,10} & r_{3,10}
\end{bmatrix}
$$

- $\{s, a, s', r\}_{i,j}$, *from state, action, to state* and *reward* of episode i at step j.
- Fix the number of experiences in the training dataset ($batch\_size$).
- Play ne episodes such that $\sum_{i=0}^{ne} ns_i > batch\_size$.
- Sample $batch\_size$ experiences from the ne episodes generated.

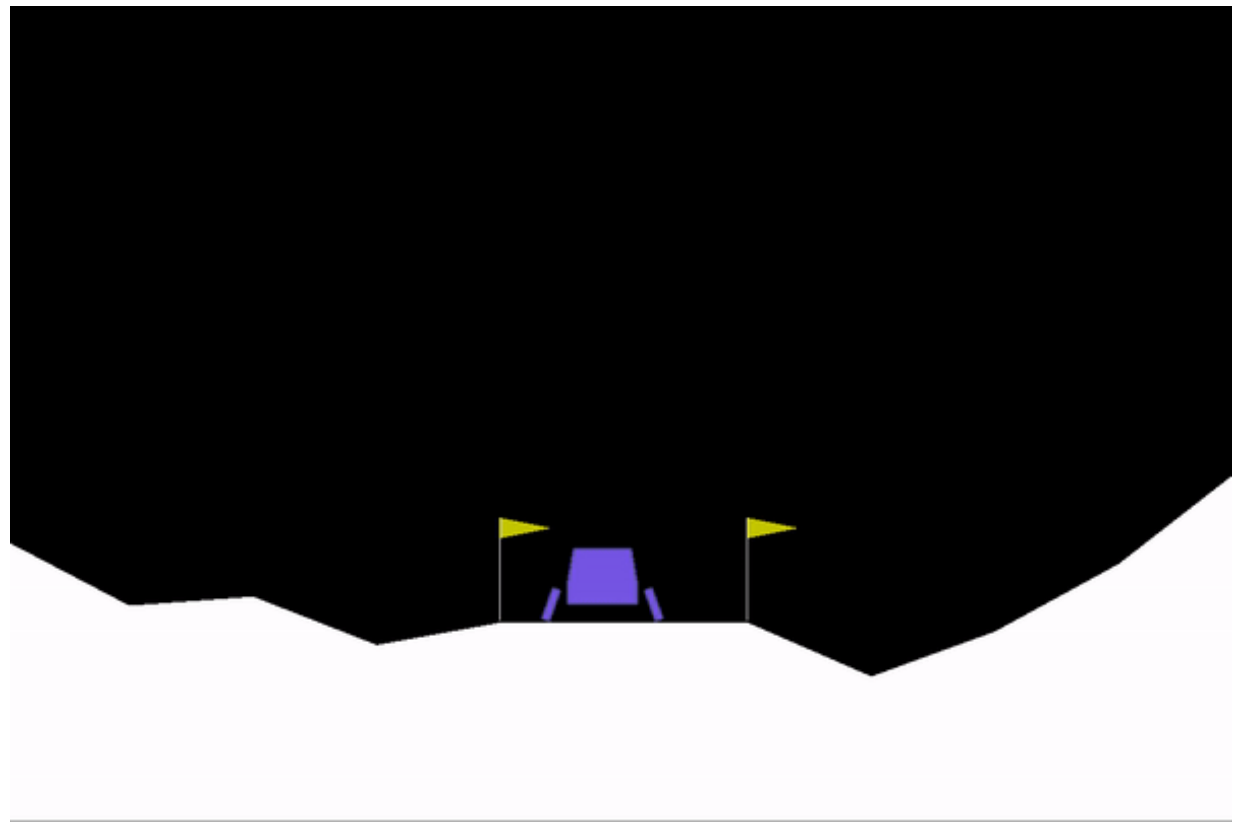
Experience replay - Cart Pole

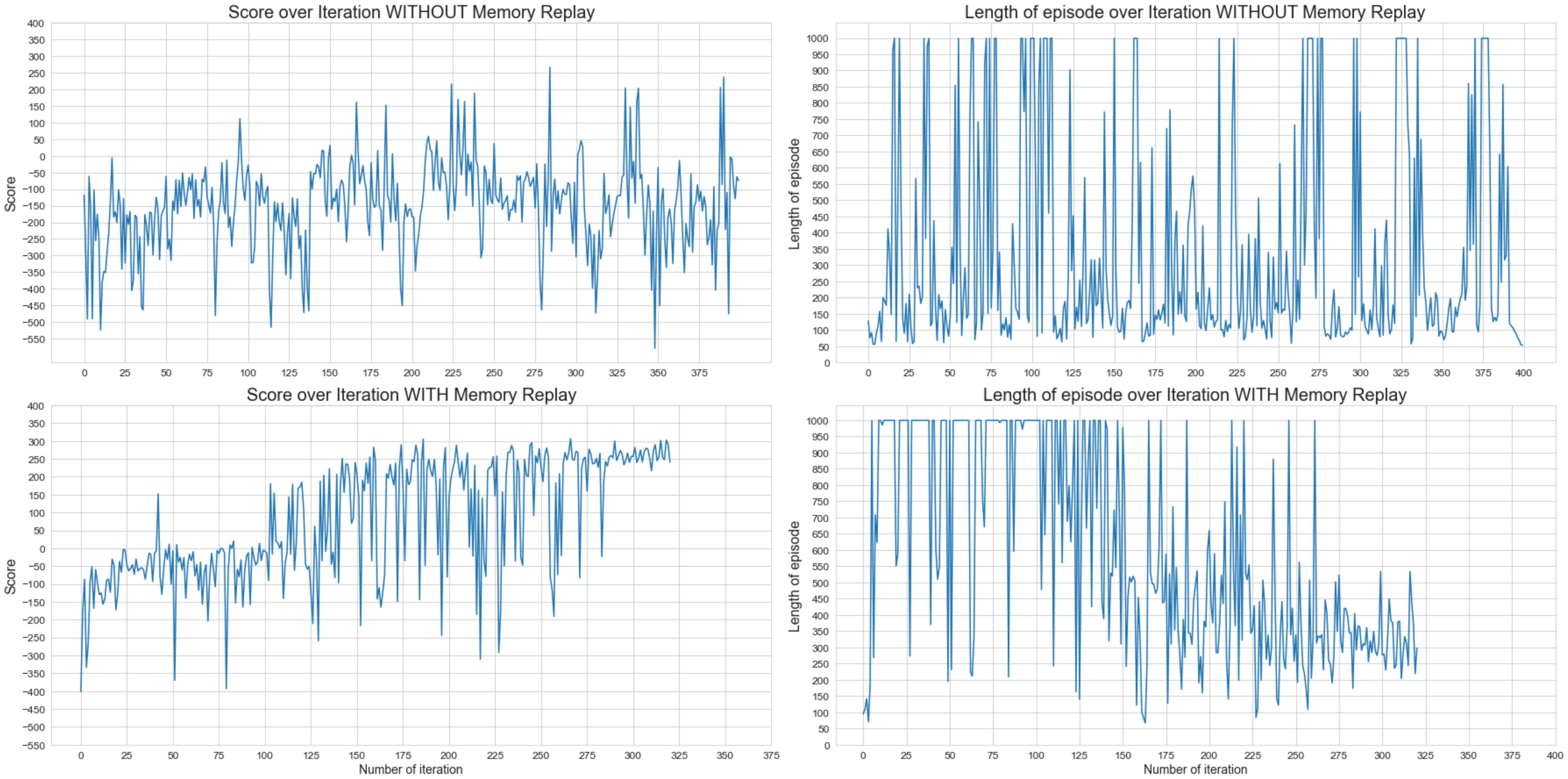






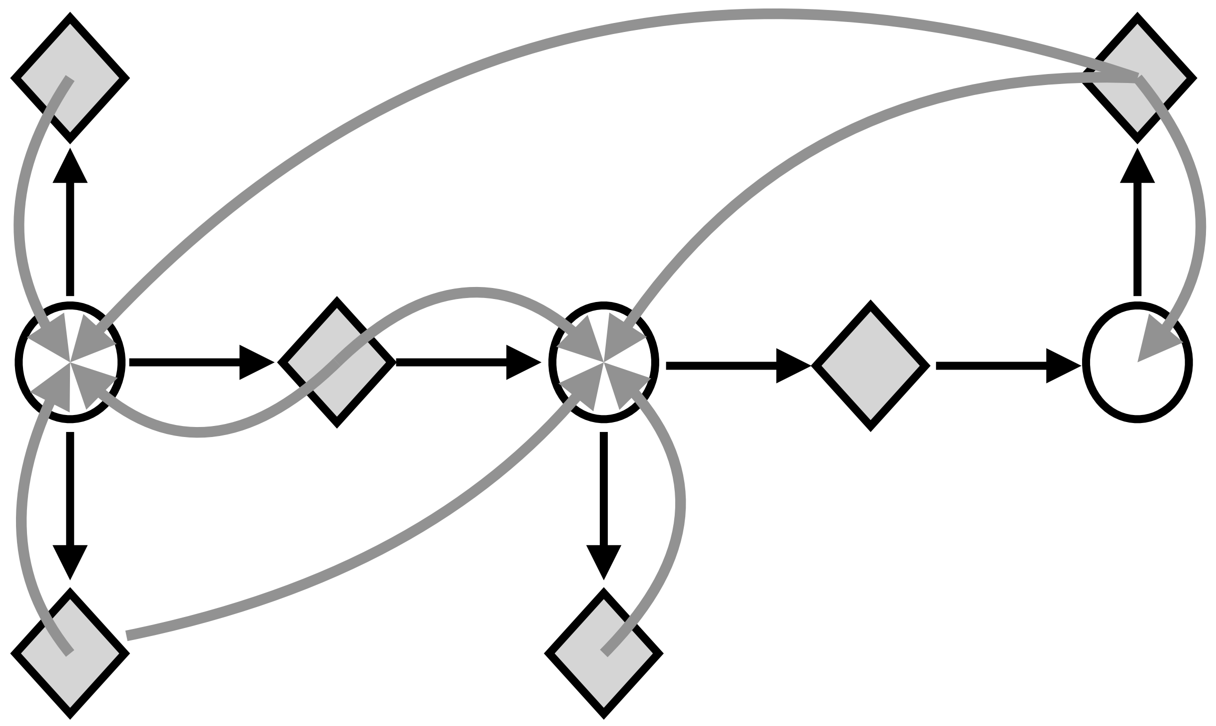
Experience replay - Lunar Landing



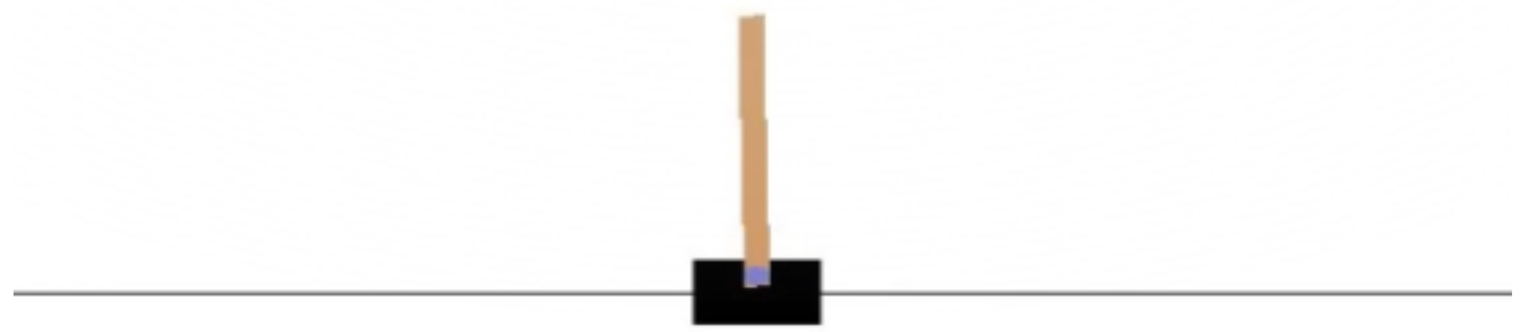


TP

- *Q_learning.ipynb :* Implement **Q-VALUE ITERATION ALGORITHM** and ***Q-LEARNING ITERATION ALGORITHM*** on the toy MDP of this presentation.



- *Deep_Q_learning_CartPole.ipynb :* Implement **Deep Q Learning** with and without Replay memory buffer on CartPole.



Target Network

Problem: The targets are unstable and depend of Q itself.

$$targets = [R(s, a, s') + \gamma \max_{a'} Q(s', a')]$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s \sim' P(s'|s,a)}[(Q_\theta(s, a) - target(s'))^2]_{\theta=\theta_k}$$

At each train iteration:

The Q values shift to get closer to the target which shift the same way.

It is chasing a **non-stationary target**

Target Network

Problem: The targets are unstable and depend of Q itself.

$$targets = [R(s,a,s') + \gamma \max_{a'} Q_{target}(s',a')]$$
$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s\sim' P(s'|s,a)}[(Q_{main\theta}(s,a) - target(s'))^2]_{\theta=\theta_k}$$

At each train iteration:

The Q values shift to get closer to the target which shift the same way.

It is chasing a **non-stationary target**

Solution: Use a Q_{target} network to generate the target (different from the Q_{main} network) .

Q_{target} network is used to generate the target: $target = [R(s,a,s') + \gamma \max_{a'} Q_{target}(s',a')]$

Q_{main} network is the network train and used to generate experiences.

Q_{target}'s weights is updated with Q_{main}'s weights from time to time

Remarks

Model Dimension

The train model Q actually take only as an input and generate all $Q(s, a)$ value:

Pacman example: $Q : s \rightarrow [Q(s, a_1), Q(s, a_2), Q(s, a_3), Q(s, a_4)]$

Hence during training, the target dataset is generate with fixed valued for not tested a and *target* for tested a.

Pacman example:

experience: $[s, a_2, s', R(s, a_2, s')]$

input: s

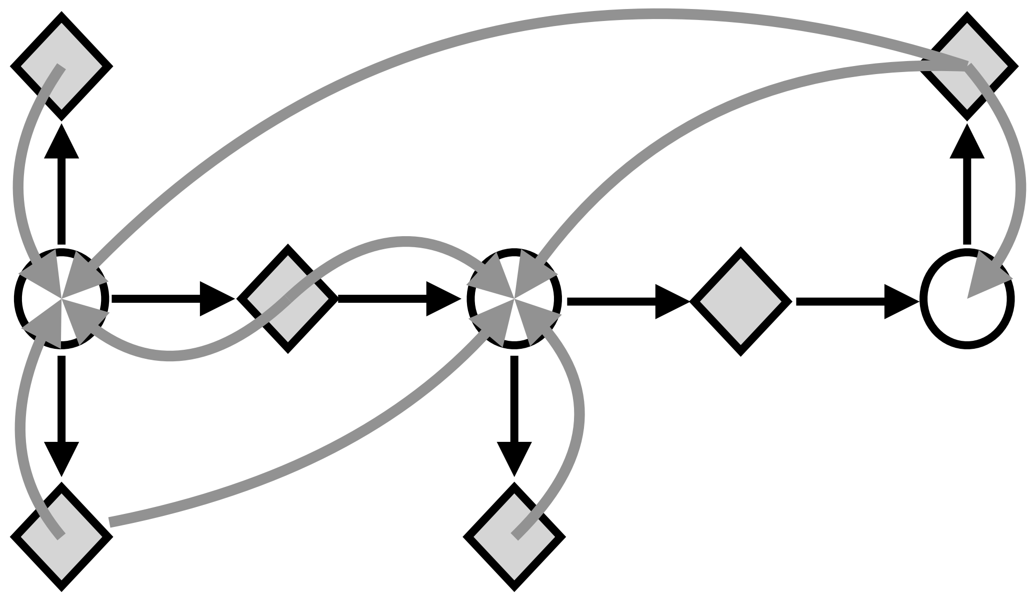
target: $$\begin{bmatrix} Q_{main}(s, a_1) \\ R(s, a_2, s') + \gamma \cdot max_{a'} Q_{target}(s', a') \\ Q_{main}(s, a_3) \\ Q_{main}(s, a_4) \end{bmatrix}$$

Deep-Q Network Mnih et al.[2015] - Pseudo code

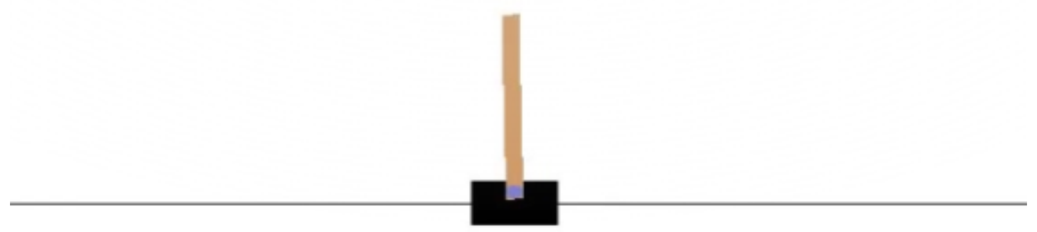
- Create Q_{main} and Q_{target} networks and initiate the replay memory M.
- Iterate while $max\_number\_episode$ or $goal\_reward$ is not achieved:
  - Play one episode - Explore randomly or Exploit with Q_{main} according to $random\_probability$.
  - Add the episode *(train data)* to M.
  - If we train more than $min\_pre\_trained\_episode$
    - Start decreasing $random\_probability$.
    - Every $train\_frequency$:
      - Sample $batch\_size$ experience from M.
      - Create target using Q_{target}
      - Train the Q_{main} network
      - Update Q_{target} weights with Q_{main} weights

TP

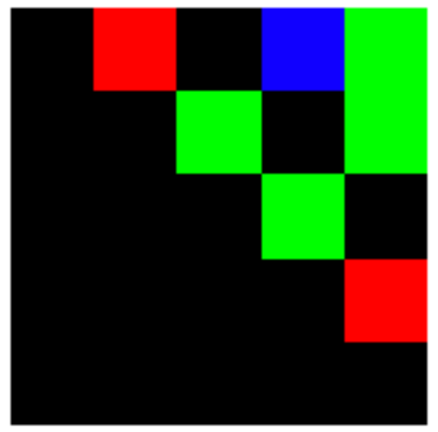
- *Q_learning.ipynb :* Implement **Q-VALUE ITERATION ALGORITHM** and ***Q-LEARNING ITERATION ALGORITHM*** on the toy MDP of this presentation.



- *Deep_Q_learning_CartPole.ipynb :* Implement **Deep Q Learning** with and without **Replay memory buffer** on CartPole.



- *Deep_Q_learning_Gridworld.ipynb :* Implement **Deep Q Learning** with **separated target network** on Gridworld.



D3QN

Deep-Q Network Improvement

Various improvement of the DQN algorithm lead to greater performance and stability.

- Dueling DQN. Wang et al. [2016]
- Doube DQN. Van Hasselt et al [2016].
- Prioritised Experience Replay. Shaul et al. [2015]

Deep-Q Network Improvement

Various improvement of the DQN algorithm lead to greater performance and stability.

- **Dueling DQN. Wang et al. [2015]**
- **Doube DQN. Van Hasselt et al [2016].**
- Prioritised Experience Replay. Shaul et al. [2015]

Dueling - Wang et al. [2016]

Main intuition

- Q-value represent how good it is to take an action a at a state s.
- It can be decomposed into two notions : $Q(s,a) = V(s) + A(s,a)$
  - The value function $V(s)$: *how good it is to be in this state?*
  - The advantage function $A(s,a)$: *how good it is to take this action in this state?*

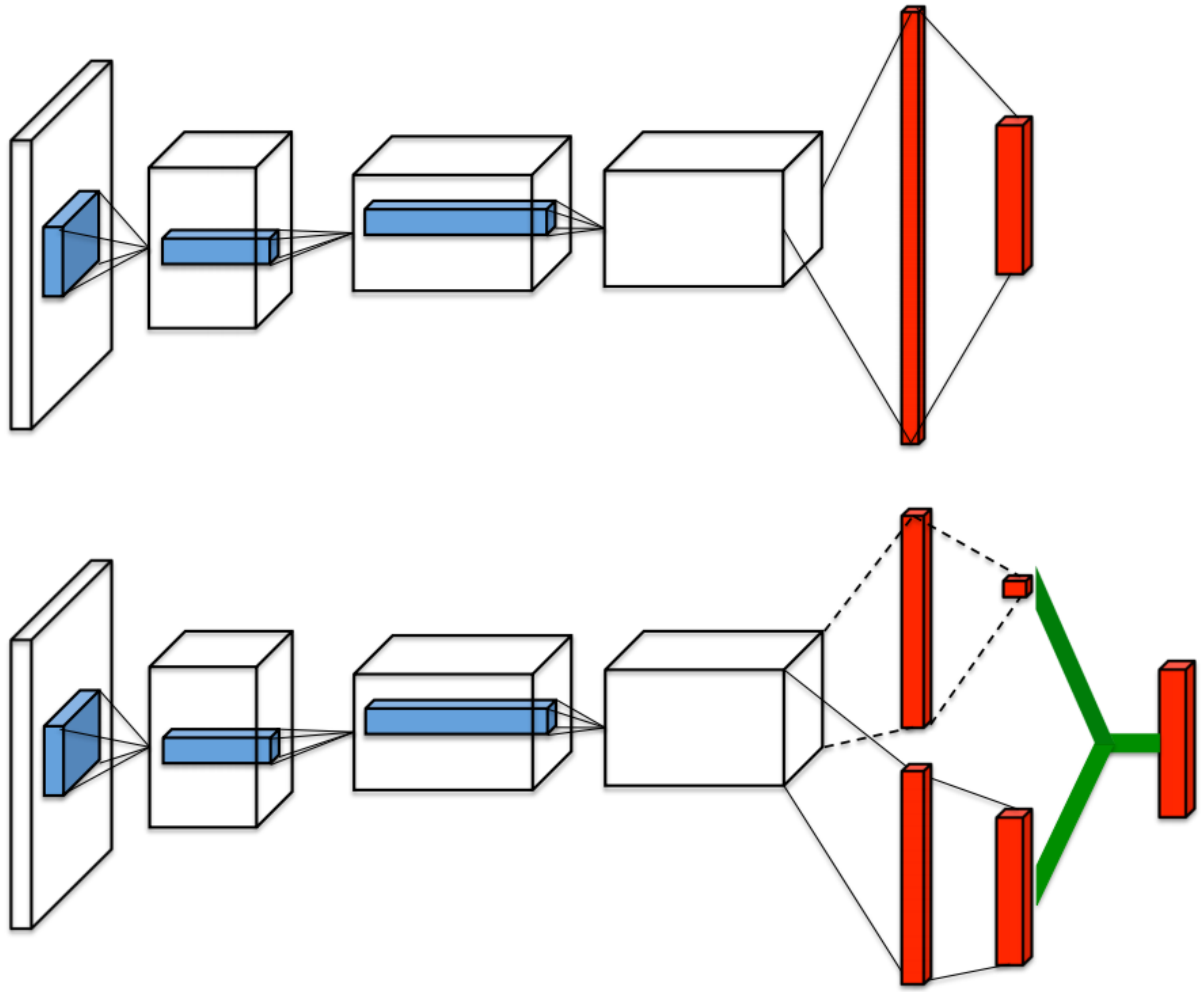
  —> It leads to poor performance.

- Let's force the Q value for the maximising action to equal V.

$$Q(s,a) = V(s) + \left(A(s,a) - max_{a' \in |\mathscr{A}|} A(s,a')\right)$$

- Which is approximate in practice by:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathscr{A}|}\sum_{a'} A(s,a')\right)$$



Double DQN

Main intuition:

DQN overestimates the Q-values of the potential action to take

Separated Target Network

$$target = R(s, a, s') + \gamma \max_{a'} Q_{target}(s', a')$$

Double DQN

Use Q_{main} to chose the action

$$target = R(s, a, s') + \gamma Q_{target}\left(s', argmax_{a'}(Q_{main}(s', a')\right)$$

Blog & Code

https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df

https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0

https://towardsdatascience.com/reinforcement-learning-tutorial-part-3-basic-deep-q-learning-186164c3bf4

https://sergioskar.github.io/Reinforcement_learning/

http://www2.econ.iastate.edu/tesfatsi/RLUsersGuide.ICAC2005.pdf

References

Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts. *Tools, and Techniques to build intelligent systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

Van Hasselt, H., Guez, A., & Silver, D. (2015). Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & Freitas, N. (2016, June). Dueling network architectures for deep reinforcement learning. In International conference on machine learning (pp. 1995-2003).