Machine Learning Summer School in Algiers

Introduction to Reinforcement Learning

Abdeslam Boularias

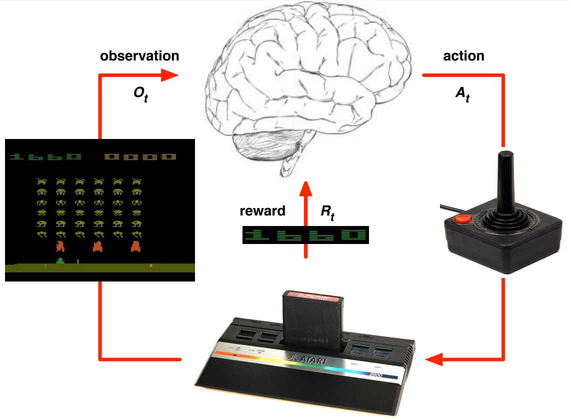
Monday, June 25, 2018

RUTGERS
THE STATE UNIVERSITY OF NEW JERSEY

What is reinforcement learning?

"a way of programming agents by reward and punishment without needing to specify how the task is to be achieved." [L. Kaelbling, M. Littman and A. Moore, 1996]

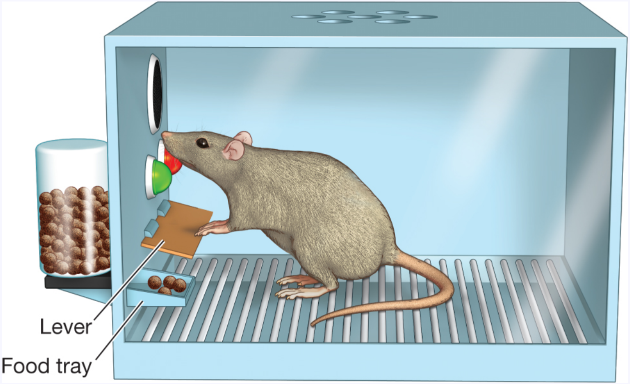
Example: Playing a video game



- Rules of the game are unknown
- Learn directly from interactive game-play
- Pick actions on joystick, see pixels and scores

from David Silver's RL course at UCL

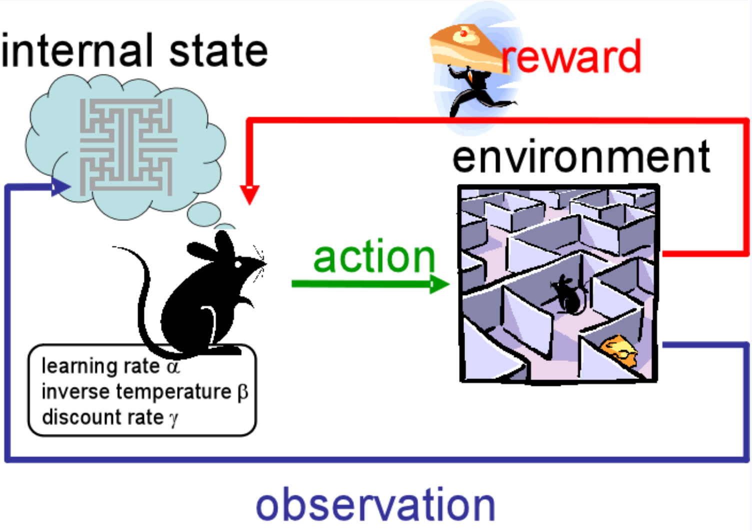
Reinforcement learning in behavioral psychology



The mouse is trained to press the lever by giving it food (positive reward) every time it presses the lever.

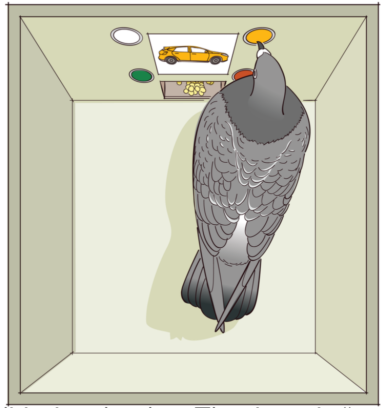
Reinforcement learning in behavioral psychology

More complex skills, such as maze navigation, can be learned from rewards.



http://www.cs.utexas.edu/ eladlieb/RLRG.html

Instrumental Conditioning



Operant conditioning chamber: The pigeon is “programmed” to click on the color of an object, by rewarding it with food.

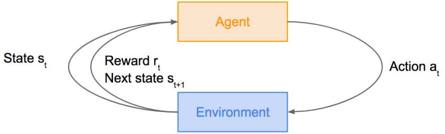
B. F. Skinner (1904-1990) a pioneer of behaviorism

- When the subject correctly performs the behavior, the chamber mechanism delivers food or another reward.
- In some cases, the mechanism delivers a punishment for incorrect or missing responses.

Reinforcement Learning

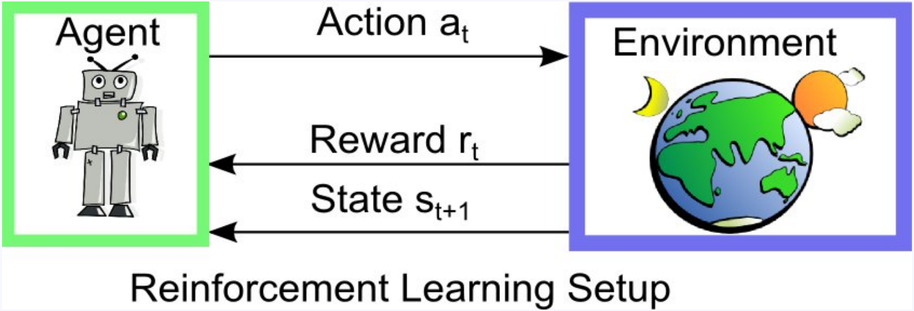
Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward



http://cs231n.stanford.edu/

Reinforcement Learning (RL)



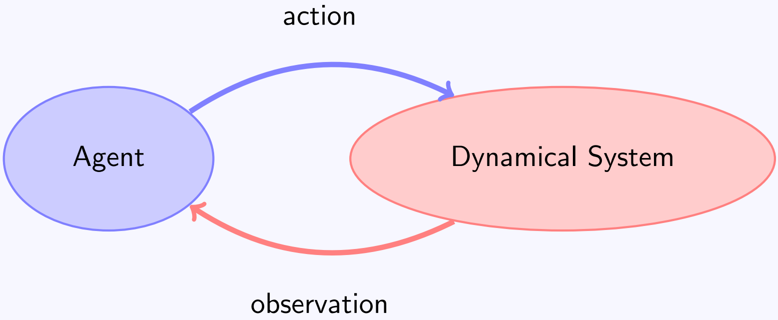
http://www.ausy.tu-darmstadt.de/Research/Research

Examples of Reinforcement Learning (RL) Applications

- Fast robotic maneuvers
- Legged locomotion
- Video games
- 3D video games
- Power grids
- Cooling Systems: DeepMind's RL Algorithms Reduce Google Data Centre Cooling Bill by 40%
- Automated Dialogue Systems (example: question-answering , Siri)
- Recommender Systems (example: online advertisements)
- Robotic manipulation

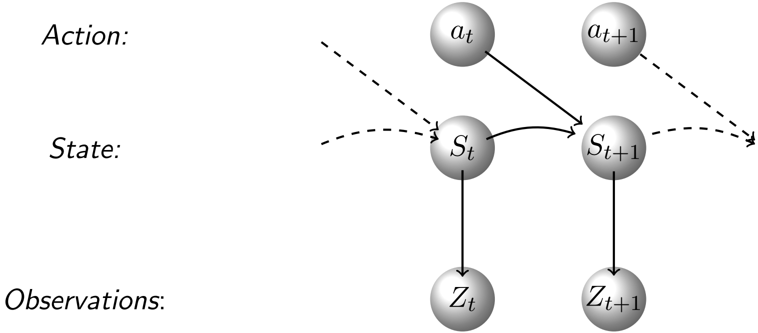
Basically, any complex dynamical system that is difficult to model analytically can be an application of RL.

Interaction between an agent and a dynamical system



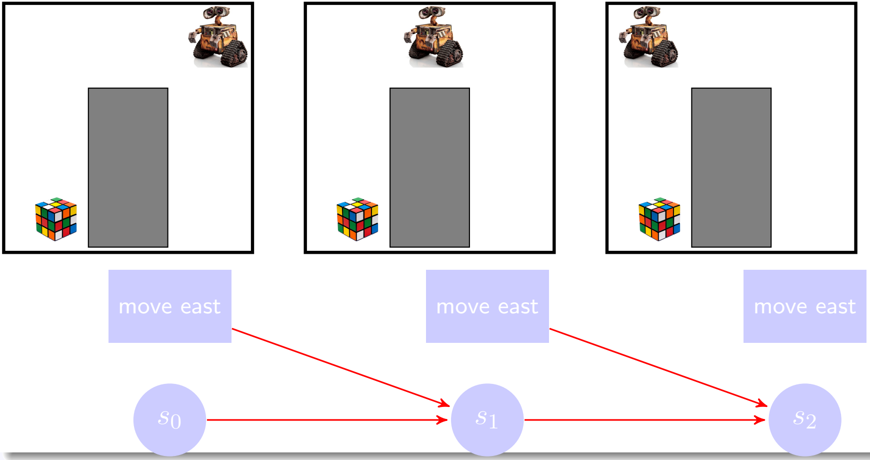
In this lecture, we consider only **fully observable** systems, where the agent always knows the current state of the system.

- **Markov Assumption**: The distribution of next states (at $t+1$) depends only on the current state and the executed action (at t).



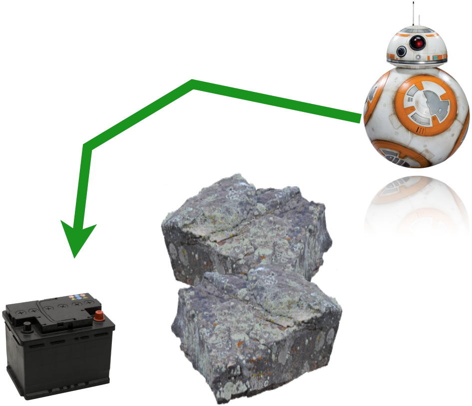
Example of decision-making problems: robot navigation

- State: position of the robot
- Actions: move east, move west, move north, move south.



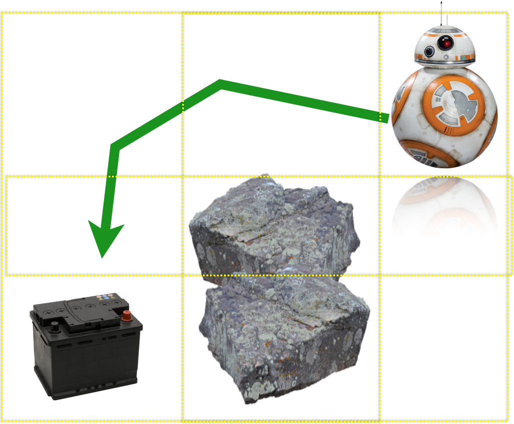
Example

Path planning: a simple sequential decision-making problem



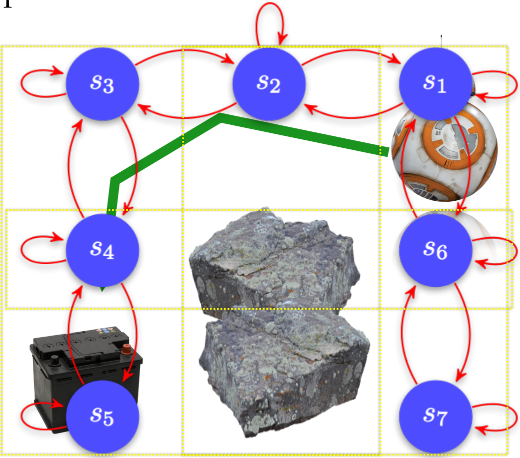
Example

Path planning: a simple sequential decision-making problem



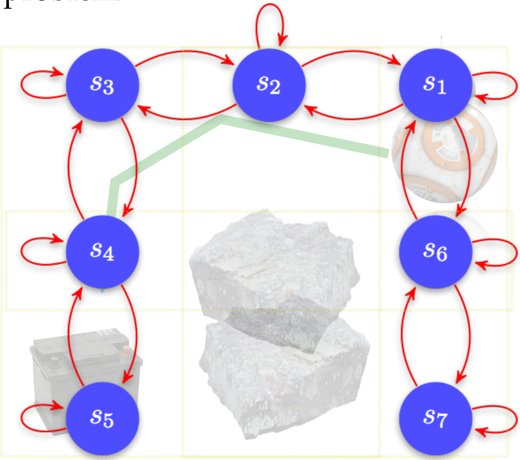
Example

Path planning: a simple sequential decision-making problem



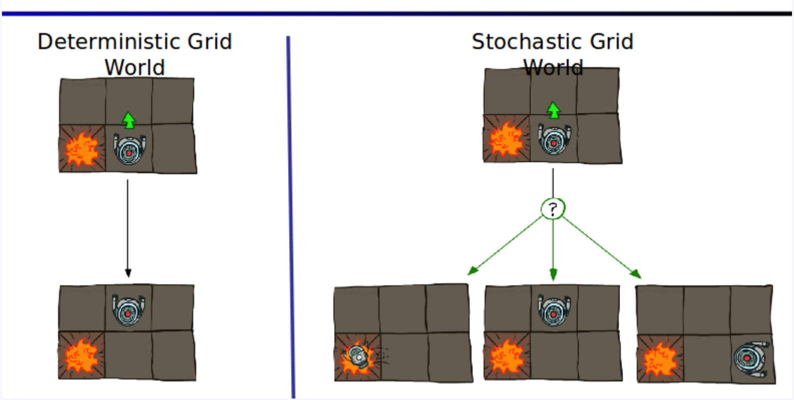
Example

Path planning: a simple sequential decision-making problem



Grid World: an example of a Markov Decision Process

Deterministic vs Stochastic Transitions



Notations

- ***S***: set of states (e.g. position and velocity of the robot)
- ***A***: set of actions (e.g. force)
- ***T***: stochastic transition function

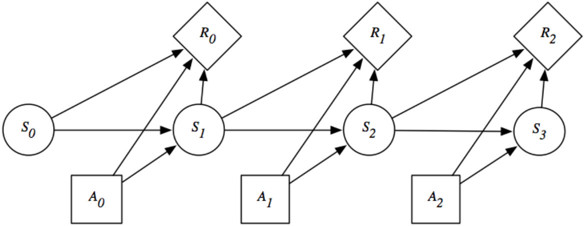
$$T(s, a, s') = Pr(\underbrace{s_{t+1}}_{\text{next state}} = s' | \underbrace{s_t}_{\text{current state}} = s, \underbrace{a_t}_{\text{current action}} = a)$$

- ***R***: reward (or cost) function

Markov Decision Process (MDP)

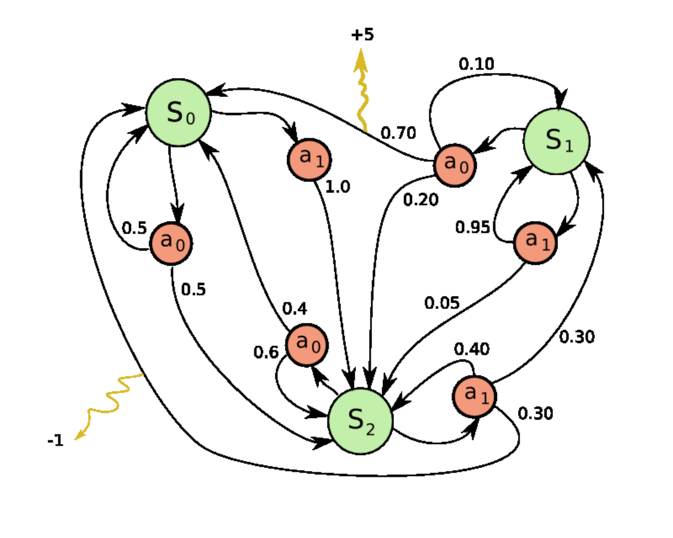
Formally, an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where:

- $\mathcal{S}$: is the space of state values.
- $\mathcal{A}$: is the space of action values.
- T: is the transition matrix.
- R: is a reward function.



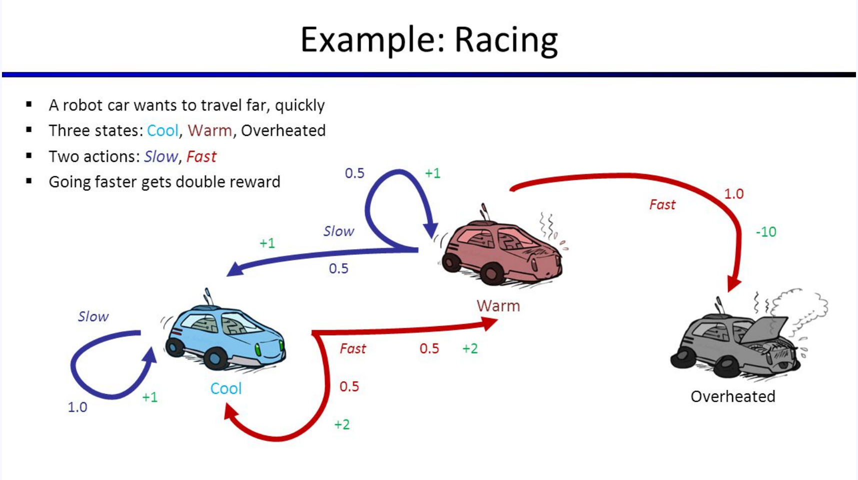
from http://artint.info

Example of a Markov Decision Process with three states and two actions



from Wikipedia

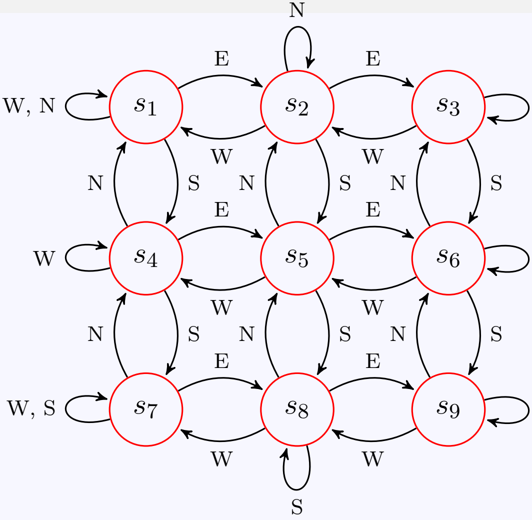
Markov Decision Process (MDP)



from Berkeley CS188

Example of a Markov Decision Process

(a) A simple navigation problem



(b) MDP representation

States set $\mathcal{S}$:

- A state is a representation of all the relevant information for predicting future states, in addition to all the information relevant for the related task.
- A state describes the configuration of the system at a given moment.
- In the example of robot navigation, the state space $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$ corresponds to the set of the robot's locations on the grid.
- The state space may be finite, countably infinite, or continuous. We will focus on models with a finite set of states. In our example, the states correspond to different positions on a discretized grid.

Actions set $\mathcal{A}$:

- The states of the system are modified by the actions executed by an agent.
- The goal is to choose actions that will steer the system to the more desirable states.
- The actions space can be finite, infinite or continuous, but we will consider only the finite case.
- In our example, the actions of the robot might be move north, move south, move east, move west, or do not move, so $\mathcal{A} = \{N, S, E, W, nothing\}$.

Transition function T:

- When an agent tries to execute an action in a given state, the action does not always lead to the same result, this is due to the fact that the information represented by the state is not sufficient for determining precisely the outcome of the actions.
- $T(s_t, a_t, s_{t+1})$ returns the probability of transitioning to state s_{t+1} after executing action a_t in state s_t.

$$T(s_t, a_t, s_{t+1}) = P(s_{t+1} \mid s_t, a_t)$$

- In our example, the actions can be either deterministic, or stochastic if the floor is slippery, and the robot might ends up in a different position while trying to move toward another one.

Markov Assumption:

$$P(\underbrace{s_{t+1}}_{future} \mid \underbrace{s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \ldots s_0, a_0}_{history}) = P(\underbrace{s_{t+1}}_{future} \mid \underbrace{s_t, a_t}_{present})$$

The current state and action have all the information needed to predict the future.

Example:

If you observe the position, velocity and acceleration of a moving vehicle at a given moment, then you could predict its position and velocity in the next few seconds without knowing its past positions, velocities or accelerations.
State = position and velocity
Action = acceleration

Open illustration from engadget.com

Reward function R:

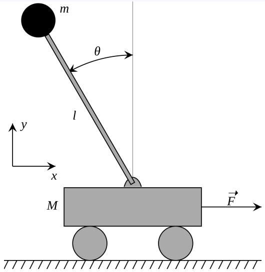
- The preferences of the agent are defined by the reward function R.
- This function directs the agent towards desirable states and keeps it away from unwanted ones. $R(s_t, a_t)$ returns a reward (or a penalty) to the agent for executing action a_t in state s_t.
- The goal of the agent is then to choose actions that maximize its cumulated reward.
- The elegance of the MDP framework comes from the possibility of modeling complex concurrent tasks by simply assigning rewards to the states.
- In our previous example, one may consider a reward of $+100$ for reaching the goal state, a -2 for any movement (consumption of energy), and a -1 for not doing anything (waste of time).

How to define the reward function R?

Examples (from David Silver's RL course at UCL)

- Fly manoeuvres in a helicopter
  - positive reward for following desired trajectory
  - negative reward for crashing
- Defeat the world champion at Backgammon
  - positive reward for winning a game
  - negative reward for losing a game
- Manage an investment portfolio
  - positive reward for each dollar in bank
- Control a power station
  - positive reward for producing power
  - reward for exceeding safety thresholds
- Make a humanoid robot walk
  - positive reward for forward motion
  - negative reward for falling over
- Play many different Atari games better than humans
  - reward for increasing/decreasing score

Examples: Cart-pole (inverted pendulum)

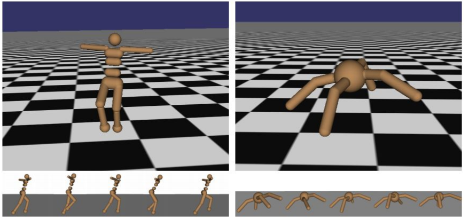


Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

http://cs231n.stanford.edu/

Examples: Robot Locomotion



Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

From OpenAI Gym (MuJoCo simulator) http://cs231n.stanford.edu/

Examples: Video Games

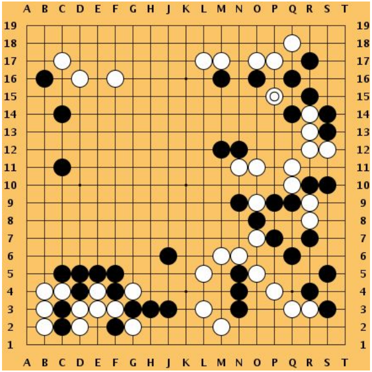
Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

Why so much interest on video games? Skills learned from games can be transferred to real-life (e.g, self-driving cars).

http://cs231n.stanford.edu/

Examples: Go game



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

http://cs231n.stanford.edu/

Given a reward function, the goal of the agent is to maximize the expected cumulated reward over some number H of steps, called the *horizon*.

$$\underbrace{(s_t, a_t, r_t), (s_{t+1}, a_{t+1}, r_{t+1}), (s_{t+2}, a_{t+2}, r_{t+2}), \ldots, (s_{t+H-1}, a_{t+H-1}, r_{t+H-1})}_{horizon}$$

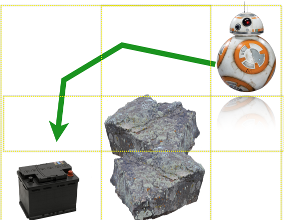
The goal of the agent is to maximize the sum of rewards
$r_t + r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_{t+H-1}$.

Finite or Infinite Horizons

- The horizon H can be either finite or infinite.
- If the horizon is finite, then the optimal actions of the agent will depend not only on the states, but also on the remaining number of steps until the end.

Example

There is a -1 reward for moving and a $+100$ reward for reaching the goal. If only 2 steps are left before the end of the episode, then it would better to do nothing and receive a cumulated reward of 0, than to move and receive a cumulative reward of -2, since the goal cannot be reached in 2 steps anyway.



- If the horizon is infinite ($H = \infty$), then the optimal actions depend only on the state. In our case, the optimal action at any step is to move toward the goal.
- A *discount factor* $\gamma \in [0, 1)$ is also used to indicate how the importance of the earned rewards decreases for every time-step delay. A reward that will be received k time-steps later is scaled down by a factor of γ^k.
- The discount factor can also be interpreted as the probability that the process continues after any step.

The goal of the agent is to maximize the sum of discounted rewards $r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \gamma^4 r_{t+4} + \gamma^5 r_{t+5} + \ldots$.

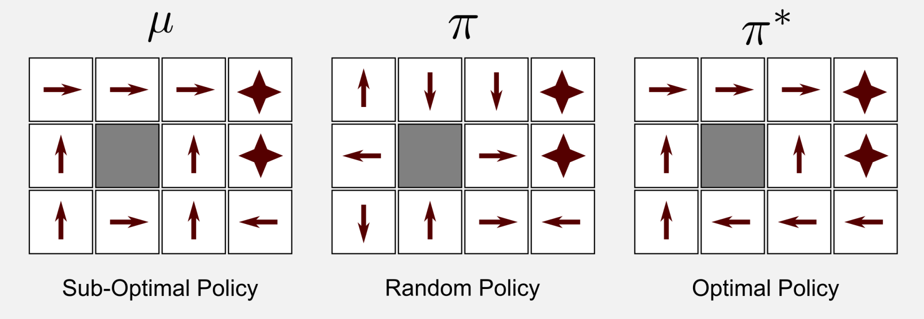
Policies

- The agent selects its actions according to a **policy** π (a *strategy*).
- A deterministic stationary policy π is a function that maps every state s into an action a.

$$\pi : \textbf{State} \rightarrow \textbf{Action}.$$

$$\pi(\textbf{State}) = \textbf{Action}.$$

Examples of Policies



Value Functions

The value function of a policy π is a function V^π that associates to each state the sum of expected rewards that the agent will receive if it starts executing policy π from that state. In other terms:

$$
\begin{aligned}
V^\pi(s) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= \textit{Sum of discounted rewards that are expected to be received} \\
&= \textit{How good is policy } \pi.
\end{aligned}
$$

where $\pi(s_t)$ is the action chosen in state s.

The value function of a policy can also be defined as:

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

The value function of a policy can also be defined as:

$$
\begin{aligned}
V^\pi(s) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= \gamma^0 R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big]
\end{aligned}
$$

The value function of a policy can also be defined as:

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{t=0}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t)) | \pi, s_0 = s\right] \\
&= \gamma^{0} R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t)) | \pi, s_0 = s\right] \\
&= R(s, \pi(s)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t)) | \pi, s_0 = s\right]
\end{aligned}
$$

The value function of a policy can also be defined as:

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{t=0}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= \gamma^{0} R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\Big[R(s_{t'}, \pi(s_{t'})) | \pi, s_0 \sim T(s, \pi(s), .)\Big]_{\text{with } t'=t-1}
\end{aligned}
$$

The value function of a policy can also be defined as:

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= \gamma^0 R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\Big[R(s_{t'}, \pi(s_{t'})) | \pi, s_0 \sim T(s, \pi(s), .)\Big]_{\text{with t'=t-1}} \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\Big[R(s_{t'}, \pi(s_{t'})) | \pi, s_0 = s'\Big]
\end{aligned}
$$

The value function of a policy can also be defined as:

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{t=0}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= \gamma^{0} R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^{t} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}_{s_t}\Big[R(s_t, \pi(s_t)) | \pi, s_0 = s\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\Big[R(s_{t'}, \pi(s_{t'})) | \pi, s_0 \sim T(s, \pi(s), .)\Big]_{\text{with } t'=t-1} \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\Big[R(s_{t'}, \pi(s_{t'})) | \pi, s_0 = s'\Big] \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V^{\pi}(s')
\end{aligned}
$$

The value function of a policy can also be defined as:

$$
\begin{aligned}
\mathbf{V}^{\pi}(s) &= \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s\right] \\
&= \gamma^0 R(s, \pi(s)) + \sum_{t=1}^{\infty} \gamma^t \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s\right] \\
&= R(s, \pi(s)) + \gamma \sum_{t=1}^{\infty} \gamma^{t-1} \mathbb{E}_{s_t}\left[R(s_t, \pi(s_t))|\pi, s_0 = s\right] \\
&= R(s, \pi(s)) + \gamma \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\left[R(s_{t'}, \pi(s_{t'}))|\pi, s_0 \sim T(s, \pi(s), .)\right]_{\text{with t'=t-1}} \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \sum_{t'=0}^{\infty} \gamma^{t'} \mathbb{E}_{s_{t'}}\left[R(s_{t'}, \pi(s_{t'}))|\pi, s_0 = s'\right] \\
&= R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \mathbf{V}^{\pi}(s')
\end{aligned}
$$

Bellman Equation

$$\begin{aligned} \mathbf{V}^{\pi}(s) &= R(s,\pi(s)) + \gamma \sum_{s' \in States} T(s,\pi(s),s')\mathbf{V}^{\pi}(s') \\ \text{value} &= \text{immediate reward} + \gamma(\text{expected value of next state}) \end{aligned}$$

This equation plays a central role in **dynamic programming**, a family of methods for solving a complex problem by breaking it down into a collection of simpler subproblems.

In dynamic programming, invented by Richard Bellman in 1957, sub-problems are nested recursively inside larger problems.

Richard Bellman (1920-1984)

Optimal policies

Bellman Equation

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}tates} T(s, \pi(s), s') V^{\pi}(s')$$

- An optimal policy π^* is one that satisfies:

$$\forall s \in \mathcal{S} : \pi^* \in \arg\max_{\pi} V^{\pi}(s)$$

- The value function of an optimal policy is called **the optimal value function**, it is defined as:

$$V^*(s) = \max_{a \in \mathcal{A}ctions} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}tates} T(s, a, s') V^*(s') \right]$$

Optimal policies

- In his seminal work on dynamic programming, Richard Bellman proved that a stationary deterministic optimal policy exists for any discounted infinite horizon MDP.
- If the value function V^π of a given policy π satisfies

$$V^\pi(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \right],$$

  then $V^\pi = V^*$ and π is an optimal policy.
- The equation above is a <u>necessary and sufficient</u> optimality condition.

In other terms, π is optimal if and only if

$$\forall s, a : \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s') \right] \leq V^\pi(s).$$

Planning

Planning: finding an optimal policy π^* given an MDP $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$.

Most of planning algorithms for MDPs fall in one of the two categories:

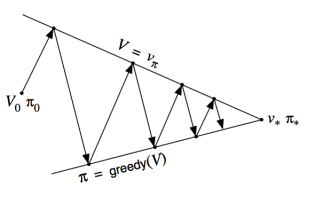
- Policy iteration
- Value iteration

Policy Iteration

- Start with a randomly chosen policy π_t at $t = 0$
- Alternate between the **policy evaluation** and the **policy improvement** operations until convergence.

$$\pi_0 \xrightarrow{\text{evaluation}} V^{\pi_0} \xrightarrow{\text{improvement}} \pi_1 \xrightarrow{\text{evaluation}} V^{\pi_1} \xrightarrow{\text{improvement}} \pi_2 \xrightarrow{\text{evaluation}}$$
$$V^{\pi_2} \xrightarrow{\text{improvement}} \pi_3 \xrightarrow{\text{evaluation}} V^{\pi_3} \xrightarrow{\text{improvement}} \pi_4 \xrightarrow{\text{evaluation}} V^{\pi_4} \xrightarrow{\text{improvement}}$$
$$\ldots\ldots \underbrace{\pi^* \xrightarrow{\text{evaluation}} V^{\pi^*} \xrightarrow{\text{improvement}} \pi^*}_{convergence} \square$$

Figure from Sutton and Barto:



Policy Iteration

- Start with a randomly chosen policy π_t at $t = 0$
- Alternate between the **policy evaluation** and the **policy improvement** operations until convergence.

Policy evaluation

- Randomly initialize the value function V_k, for $k = 0$.
- Repeat the operation:

$$\forall s \in States : V_{k+1}(s) \leftarrow R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$ for a predefined error threshold ϵ.

Policy Iteration

- Start with a randomly chosen policy π_t at $t = 0$
- Alternate between the **policy evaluation** and the **policy improvement** operations until convergence.

Policy improvement

Find a *greedy* policy π_{t+1} given the value function V_k (computed in the policy evaluation phase):

$$\forall s \in \mathcal{S} : \pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}ctions} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}tates} T(s,a,s') V_k(s') \right]$$

The policy iteration process stops when $\pi_t = \pi_{t-1}$, in which case π_t is an optimal policy, i.e. $\pi_t = \pi^*$.

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;

```
/* Initialization */ ;
```

$t = 0$, $k = 0$;

$\forall s \in \mathcal{S}$: Initialize $\pi_t(s)$ with an arbitrary action;

$\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;

repeat

```
/* Policy evaluation */;
```

repeat

$\forall s \in \mathcal{S} : V_{k+1}(s) \leftarrow R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$;

$k \leftarrow k + 1$;

until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;

```
/* Policy improvement */;
```

$\forall s \in \mathcal{S} : \pi_{t+1}(s) \leftarrow \arg\max_{a \in \mathcal{A}} \left[R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_k(s') \right]$;

$t \leftarrow t + 1$;

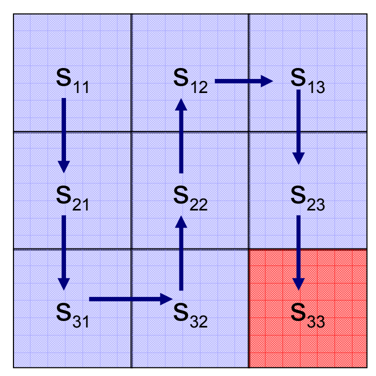
until $\pi_t = \pi_{t-1}$;

$\pi^* = \pi_t$;

Output: An optimal policy π^*;

Example

- State space $\mathcal{S} = \{s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, s_{31}, s_{32}, s_{33}\}$
- Action space $\mathcal{A} = \{\leftarrow, \rightarrow, \uparrow, \downarrow, \text{do nothing}\}$
- Deterministic transition function
- Reward function $\forall a : R(s_{33}, a) = 1$, $\forall a, \forall s \neq s_{33} : R(s, a) = 0$
- Discount factor $\gamma = 0.9$.

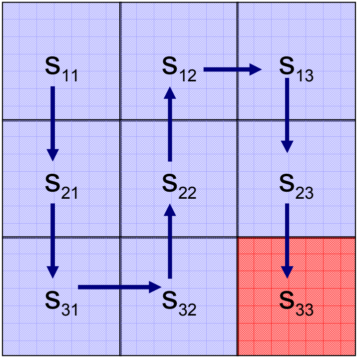


Initial policy

Example

Let's perform the policy evaluation on the initial policy

State	V_0
s_{11}	0
s_{12}	0
s_{13}	0
s_{21}	0
s_{22}	0
s_{23}	0
s_{31}	0
s_{32}	0
s_{33}	0



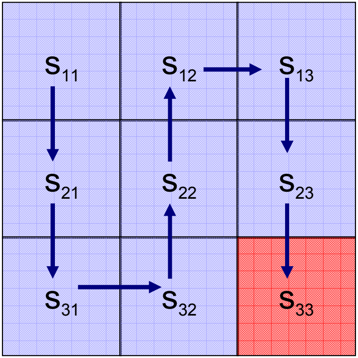
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1
s_{11}	0	$0 + 0.9 \times 0$
s_{12}	0	$0 + 0.9 \times 0$
s_{13}	0	$0 + 0.9 \times 0$
s_{21}	0	$0 + 0.9 \times 0$
s_{22}	0	$0 + 0.9 \times 0$
s_{23}	0	$0 + 0.9 \times 0$
s_{31}	0	$0 + 0.9 \times 0$
s_{32}	0	$0 + 0.9 \times 0$
s_{33}	0	$1 + 0.9 \times 0$



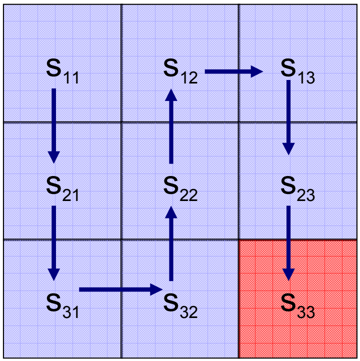
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1
s_{11}	0	0
s_{12}	0	0
s_{13}	0	0
s_{21}	0	0
s_{22}	0	0
s_{23}	0	0
s_{31}	0	0
s_{32}	0	0
s_{33}	0	1



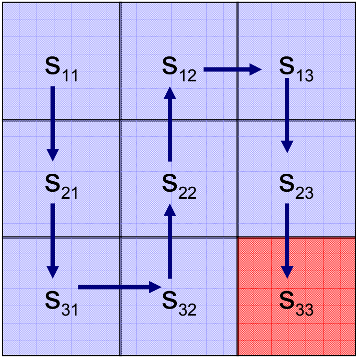
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2
s_{11}	0	0	$0 + 0.9 \times 0$
s_{12}	0	0	$0 + 0.9 \times 0$
s_{13}	0	0	$0 + 0.9 \times 0$
s_{21}	0	0	$0 + 0.9 \times 0$
s_{22}	0	0	$0 + 0.9 \times 0$
s_{23}	0	0	$0 + 0.9 \times 1$
s_{31}	0	0	$0 + 0.9 \times 0$
s_{32}	0	0	$0 + 0.9 \times 0$
s_{33}	0	1	$1 + 0.9 \times 1$



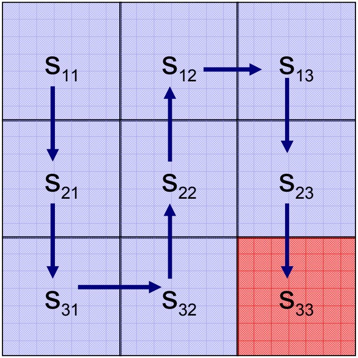
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2
s_{11}	0	0	0
s_{12}	0	0	0
s_{13}	0	0	0
s_{21}	0	0	0
s_{22}	0	0	0
s_{23}	0	0	0.9
s_{31}	0	0	0
s_{32}	0	0	0
s_{33}	0	1	1.9



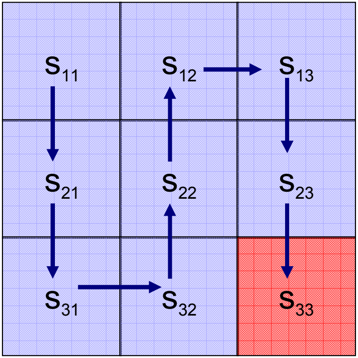
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3
s_{11}	0	0	0	$0 + 0.9 \times 0$
s_{12}	0	0	0	$0 + 0.9 \times 0$
s_{13}	0	0	0	$0 + 0.9 \times 0.9$
s_{21}	0	0	0	$0 + 0.9 \times 0$
s_{22}	0	0	0	$0 + 0.9 \times 0$
s_{23}	0	0	0.9	$0 + 0.9 \times 1.9$
s_{31}	0	0	0	$0 + 0.9 \times 0$
s_{32}	0	0	0	$0 + 0.9 \times 0$
s_{33}	0	1	1.9	$1 + 0.9 \times 1.9$



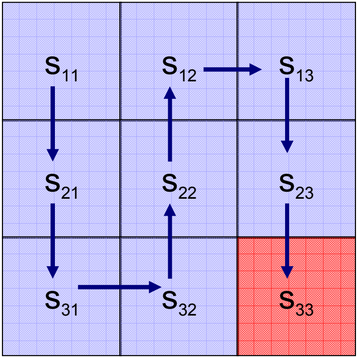
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3
s_{11}	0	0	0	0
s_{12}	0	0	0	0
s_{13}	0	0	0	0.81
s_{21}	0	0	0	0
s_{22}	0	0	0	0
s_{23}	0	0	0.9	1.71
s_{31}	0	0	0	0
s_{32}	0	0	0	0
s_{33}	0	1	1.9	2.71



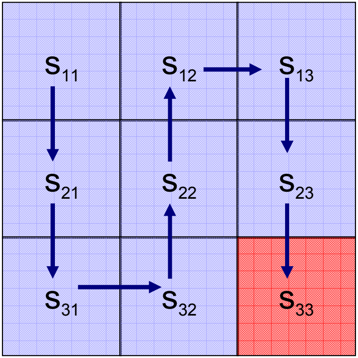
Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Example

Let's perform the policy evaluation on the initial policy

State	V_0	V_1	V_2	V_3	$\dots$	V_{1000}
s_{11}	0	0	0	0		4.3
s_{12}	0	0	0	0		7.3
s_{13}	0	0	0	0.81		8.1
s_{21}	0	0	0	0		4.8
s_{22}	0	0	0	0		6.6
s_{23}	0	0	0.9	1.71		9
s_{31}	0	0	0	0		5.3
s_{32}	0	0	0	0		5.9
s_{33}	0	1	1.9	2.71		10

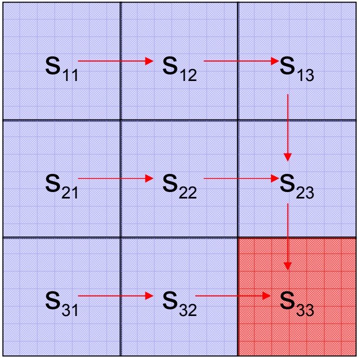


Initial policy

$$\forall s \in \mathcal{S} : V_{k+1}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi_t(s), s') V_k(s')$$

Now, we improve the previous policy based on the calculated values

State	V_0	V_1	V_2	V_3	$\dots$	V_{1000}
s_{11}	0	0	0	0		4.3
s_{12}	0	0	0	0		7.3
s_{13}	0	0	0	0.81		8.1
s_{21}	0	0	0	0		4.8
s_{22}	0	0	0	0		6.6
s_{23}	0	0	0.9	1.71		9
s_{31}	0	0	0	0		5.3
s_{32}	0	0	0	0		5.9
s_{33}	0	1	1.9	2.71		10



Improved policy

$$\forall s \in \mathcal{S} : \pi_{t+1}(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right].$$

Repeat policy evaluation with the new policy π_{t+1}. Stop if $\pi_{t+1} = \pi_t$.

Value iteration can be written as a simple backup operation:

$$\forall s \in \mathcal{S} : V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

This operation is repeated until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$, in which case the optimal policy is simply the greedy policy with respect to the value function V_k

$$\forall s \in \mathcal{S} : \pi^*(s) = \arg\max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \right]$$

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$;
$k = 0$;
$\forall s \in \mathcal{S}$: Initialize $V_k(s)$ with an arbitrary value;
repeat
$\quad \forall s \in \mathcal{S} : V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \Big]$;
$\quad k \leftarrow k + 1$;
until $\forall s \in \mathcal{S} : |V_k(s) - V_{k-1}(s)| < \epsilon$;
$\forall s \in \mathcal{S} : \pi^*(s) = \arg\max_{a \in \mathcal{A}} \Big[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_k(s') \Big]$;
Output: An optimal policy π^*;

Algorithm 2: The value iteration algorithm.

Learning with Markov Decision Processes

How can we find an optimal policy when we do not know the transition function T?

Reinforcement Learning (RL)

- Generally refers to the problem of finding an optimal policy π^* for an MDP with unknown transition function T.
- The agent learns, the best actions from experience, by acting and observing the received rewards, i.e. by *trial-and-error*.

image credit: Remi Munos

Model-based vs Model-free RL

Model-based Approach to Reinforcement Learning

- Collect data: $Data = \{(s_t, a_t, s_{t+1}), \text{for } t = 0, \dots N\}$
- Estimate the transition function as: for any states s and s', and action a

$$T(s,a,s') = P(s'|s,a) \approx \frac{\text{Number of times } (s,a,s') \text{ appears in } Data}{\text{Number of times } (s,a,_{\text{anything}}) \text{ appears in } Data}$$

  where the denominator is the total number of times that action a was executed in state s in the data, regardless of the next state.
- These estimates converge to the true model T if $\mathcal{S}$ and $\mathcal{A}$ are finite.
- Find an optimal policy using the Policy Iteration or the Value Iteration algorithms with the learned model T.

Model-free Approach to Reinforcement Learning

Learn the policy directly from the rewards, without learning the transition function

- It is not necessary to learn a model
- More robust to modeling errors
- Much simpler than model-based approaches
- Typically requires more data for training

Q-value

Before presenting some learning algorithms, we will first need to introduce the *Q-value function*.

Q-value

A *Q-value* is the expected sum of rewards that an agent will receive if it executes action a in state s then follows a policy π for the remaining steps.

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^\pi(s')$$

a can be any action, it is not necessarily $\pi(s)$.

Value Function and Q-Value Function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state?
The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

How good is a state-action pair?
The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

http://cs231n.stanford.edu/

The Q-learning algorithm

We'd like to do Q-value updates to each Q-state:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- But can't compute this update without knowing T, R

Instead, compute average as we go

- Receive a sample transition (s,a,r,s')
- This sample suggests

$$Q(s,a) \approx r + \gamma \max_{a'} Q(s',a')$$

- But we want to average over results from (s,a) (Why?)
- So keep a running average

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha) \left[r + \gamma \max_{a'} Q(s',a') \right]$$

α is just any number between 0 and 1 that is decreased over time.

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

Input: An MDP model $\langle \mathcal{S}, \mathcal{A}, R \rangle$ with unknown transition function;
$t = 0$, s_0 is an initial state;
$\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$: Initialize $Q^t(s, a)$ with an arbitrary value;
repeat

$\pi(s_t) = \arg\max_{a \in \mathcal{A}} Q^t(s_t, a)$;
Choose action a_t as $\pi(s_t)$ with probability $1 - \epsilon_t$ (for exploitation), and as a random action (for exploration) with probability ϵ_t;
Execute action a_t and observe the received reward $R(s_t, a_t)$ and the next state s_{t+1};

$$Q^{t+1}(s_t, a_t) = (1 - \alpha_t) Q^t(s_t, a_t) + \alpha_t \left[R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q^t(s_{t+1}, a') \right]$$

$t \leftarrow t + 1$;

until *the end of learning*;
Output: A learned policy π;

Algorithm 3: The Q-learning algorithm.

Alternative Formulation of the Update Equation

$$Q^{t+1}(s_t, a_t) = Q^t(s_t, a_t) + \alpha_t \Big[\underbrace{\overbrace{R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q^t(s_{t+1}, a')}^{\text{observed value}} - \overbrace{Q^t(s_t, a_t)}^{\text{predicted value}}}_{\text{Temporal Difference (TD)}} \Big]$$

New Value = Old Value + (learning rate) * (Observed Value - Predicted Value)

Convergence conditions of tabular Q-learning (discrete states and actions)

Robbins-Monro conditions for the learning rate

- $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ and,
- $\sum_{t=0}^{\infty} \alpha_t = \infty$

In other terms

- Learning rate α_t decreases over time,
- but not too fast.

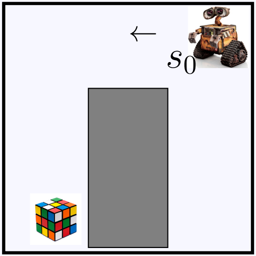
The exploration probability ϵ_t should be non-zero.

Example of good α_t and ϵ_t

$$\alpha_t = \frac{1}{t}, \qquad \epsilon_t = \frac{1}{\sqrt{t}}$$

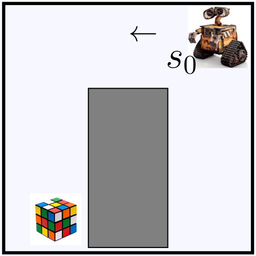
Suppose

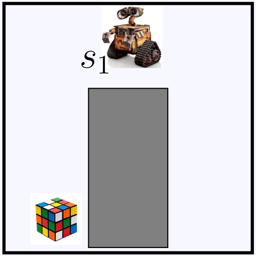
- The agent selects action *move left* in state s_0

time-step 0: 

Suppose

- The agent selects action *move left* in state s_0
- The agent gets a reward $R(s_0, \textit{move left}) = 10$ and moves to state s_1

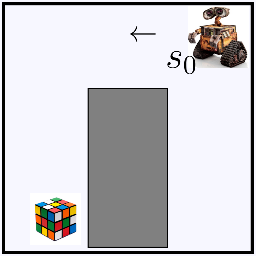
time-step 0: 

time-step 1: 

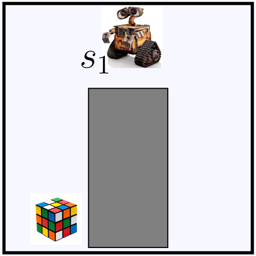
Suppose

- The agent selects action *move left* in state s_0
- The agent gets a reward $R(s_0, \textit{move left}) = 10$ and moves to state s_1
- The old Q-value of $(s_0, \textit{move left})$ is $Q(s_0, \textit{move left}) = 3$

time-step 0:



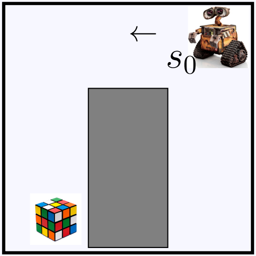
time-step 1:



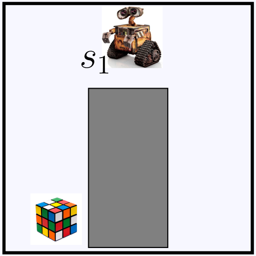
Suppose

- The agent selects action *move left* in state s_0
- The agent gets a reward $R(s_0, \textit{move left}) = 10$ and moves to state s_1
- The old Q-value of $(s_0, \textit{move left})$ is $Q(s_0, \textit{move left}) = 3$
- The max Q-value in state s_1 is $Q(s_1, \textit{move up}) = 7$

time-step 0:



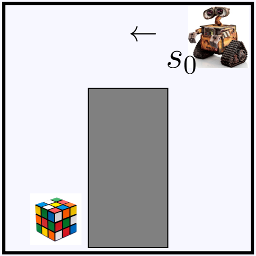
time-step 1:



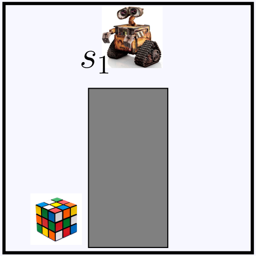
Suppose

- The agent selects action *move left* in state s_0
- The agent gets a reward $R(s_0, \textit{move left}) = 10$ and moves to state s_1
- The old Q-value of $(s_0, \textit{move left})$ is $Q(s_0, \textit{move left}) = 3$
- The max Q-value in state s_1 is $Q(s_1, \textit{move up}) = 7$
- The discount factor $\gamma = 0.95$
- The learning rate $\alpha = 0.1$

time-step 0:



time-step 1:



Suppose

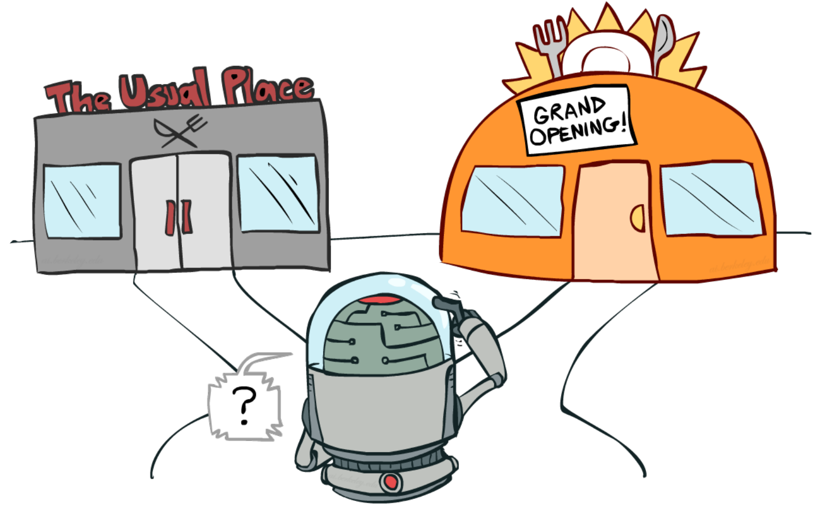
- The agent selects action *move left* in state s_0
- The agent gets a reward $R(s_0, \textit{move left}) = 10$ and moves to state s_1
- The old Q-value of $(s_0, \textit{move left})$ is $Q(s_0, \textit{move left}) = 3$
- The max Q-value in state s_1 is $Q(s_1, \textit{move up}) = 7$
- The discount factor $\gamma = 0.95$
- The learning rate $\alpha = 0.1$

Then the Q-value is updated as

$$Q(s_0, \textit{move left}) \leftarrow Q(s_0, \textit{move left}) + \alpha \Big[R(s_0, \textit{move left}) + \gamma Q(s_1, \textit{move up}) - Q(s_0, \textit{move left}) \Big]$$

$$Q(s_0, \textit{move left}) \leftarrow 3 + 0.1 * \Big[10 + 0.95 * 7 - 3 \Big] = 4.365$$

Exploration vs. Exploitation



https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

Several schemes for forcing exploration

- Simplest: random actions (ε-greedy)
  - Every time step, flip a coin
  - With (small) probability ε, act randomly
  - With (large) probability $1-\varepsilon$, act on current policy

- Problems with random actions?
  - You do eventually explore the space, but keep thrashing around once learning is done
  - One solution: lower ε over time
  - Another solution: exploration functions

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

How to Explore?

When to explore?

- Random actions: explore a fixed amount
- Better idea: explore areas whose badness is not (yet) established, eventually stop exploring

Exploration function

- Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$

Regular Q-Update: $Q(s, a) \leftarrow_\alpha R(s, a, s') + \gamma \max_{a'} Q(s', a')$

Modified Q-Update: $Q(s, a) \leftarrow_\alpha R(s, a, s') + \gamma \max_{a'} f(Q(s', a'), N(s', a'))$

- Note: this propagates the “bonus” back to states that lead to unknown states as well!

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

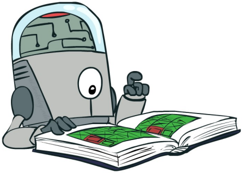
Basic Q-Learning keeps a table of all q-values

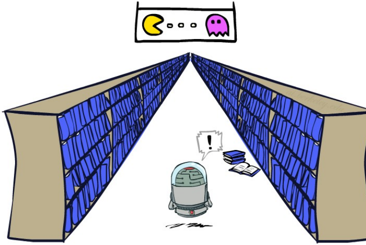
In realistic situations, we cannot possibly learn about every single state!

- Too many states to visit them all in training
- Too many states to hold the q-tables in memory

Instead, we want to generalize:

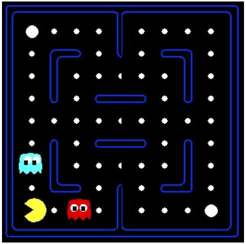
- Learn about some small number of training states from experience
- Generalize that experience to new, similar situations
- This is a fundamental idea in machine learning, and we'll see it over and over again



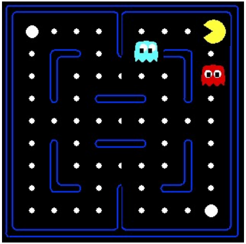


https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

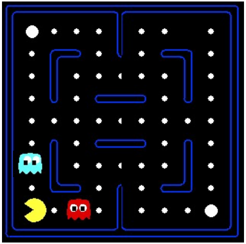
Let's say we discover through experience that this state is bad:



In naïve q-learning, we know nothing about this state:



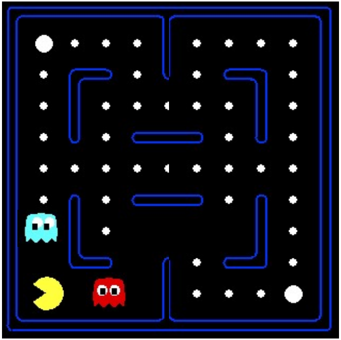
Or even this one!



https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

Solution: describe a state using a vector of features (properties)

- Features are functions from states to real numbers (often 0/1) that capture important properties of the state
- Example features:
  - Distance to closest ghost
  - Distance to closest dot
  - Number of ghosts
  - 1 / (dist to dot)^2
  - Is Pacman in a tunnel? (0/1)
  - …… etc.
  - Is it the exact state on this slide?
- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

Advantage: our experience is summed up in a few powerful numbers

Disadvantage: states may share features but actually be very different in value!

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

If everything is summed up in weights w, how can we learn them?

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

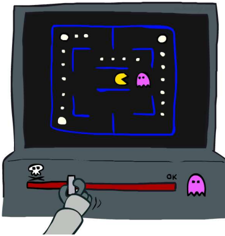
Q-learning with linear Q-functions:

transition $= (s, a, r, s')$

difference $= \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha$ [difference] Exact Q's

$w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$ Approximate Q's



Intuitive interpretation:

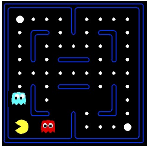
- Adjust weights of active features
- E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

Formal justification: online least squares

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$

s

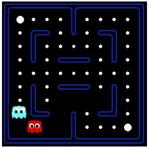


$f_{DOT}(s, \text{NORTH}) = 0.5$

$f_{GST}(s, \text{NORTH}) = 1.0$

$a = \text{NORTH}$
$r = -500$

s'



$Q(s, \text{NORTH}) = +1$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

$Q(s', \cdot) = 0$

$\text{difference} = -501$

$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$

$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$

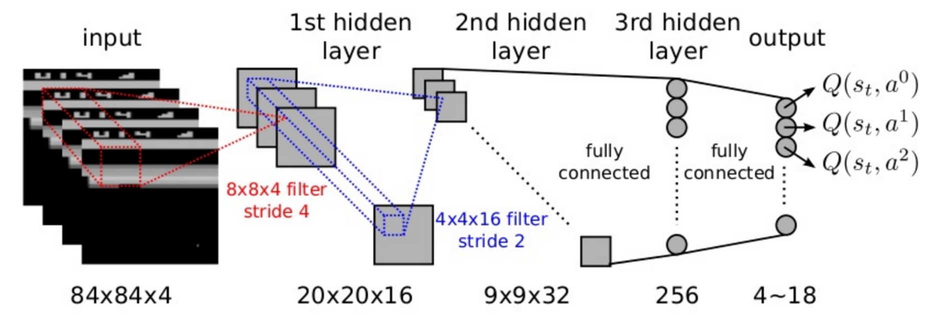
$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

[Demo: approxima

Where the learning rate α is set in this example to $\approx 1/250$

https://courses.cs.washington.edu/courses/cse473/17wi/slides/11-rl2.pdf

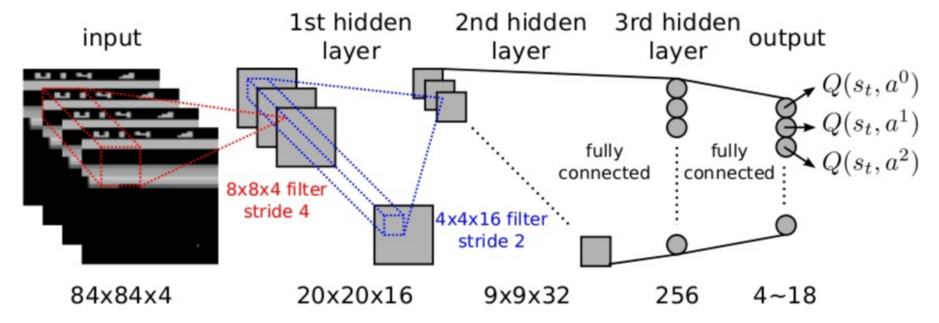
Deep Q Learning (DQN, Mnih et al., 2013)



Loss Function

$$min_w \sum_{t=0}^{N} \Bigg(\underbrace{\overbrace{R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q_w(s_{t+1}, a')}^{\text{observed value}} - \overbrace{Q_w(s_t, a_t)}^{\text{predicted value}}}_{\text{Temporal Difference (TD)}} \Bigg)^2$$

Dueling Network Architectures for Deep RL (Wang et al., 2015)



Loss Function

$$min_w \sum_{t=0}^{N} \Bigg(\underbrace{\overbrace{R(s_t, a_t) + \gamma \max_{a' \in \mathcal{A}} Q'_w(s_{t+1}, a')}^{\text{observed value}} - \overbrace{Q_w(s_t, a_t)}^{\text{predicted value}}}_{\text{Temporal Difference (TD)}} \Bigg)^2$$

Questions