REINFORCEMENT LEARNING

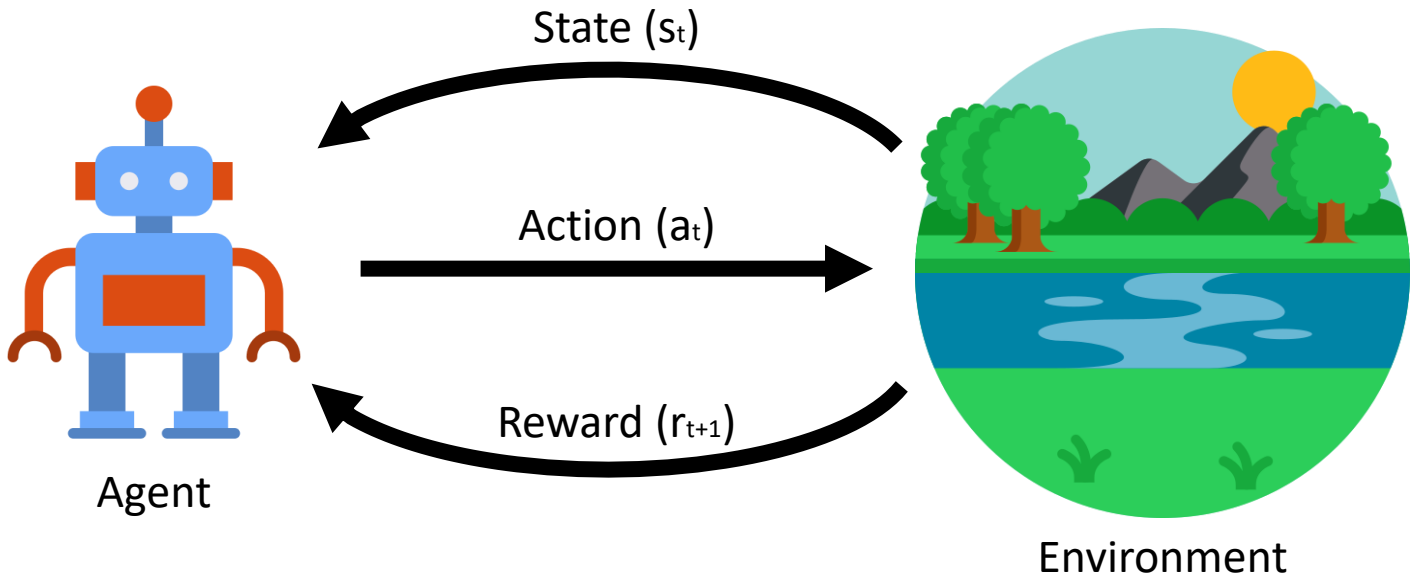
Juan Jesús Roldán Gómez

- Web: www.jjrg.org
- Email: juan.roldan@uam.es
- Office: EPS-UAM, B-321

REINFORCEMENT LEARNING

Reinforcement Learning

Machine learning method in which an agent learns through the interaction with its environment.



Basic problem:

1. The **agent** observes the **state** of the **environment**.
2. The **agent** performs an **action**.
3. The **state** of the **environment** changes because of this **action**.
4. The **agent** receives a **reward**.

The agent aims at learning to choose the best actions in every state to maximize the sum of rewards.

Reinforcement Learning

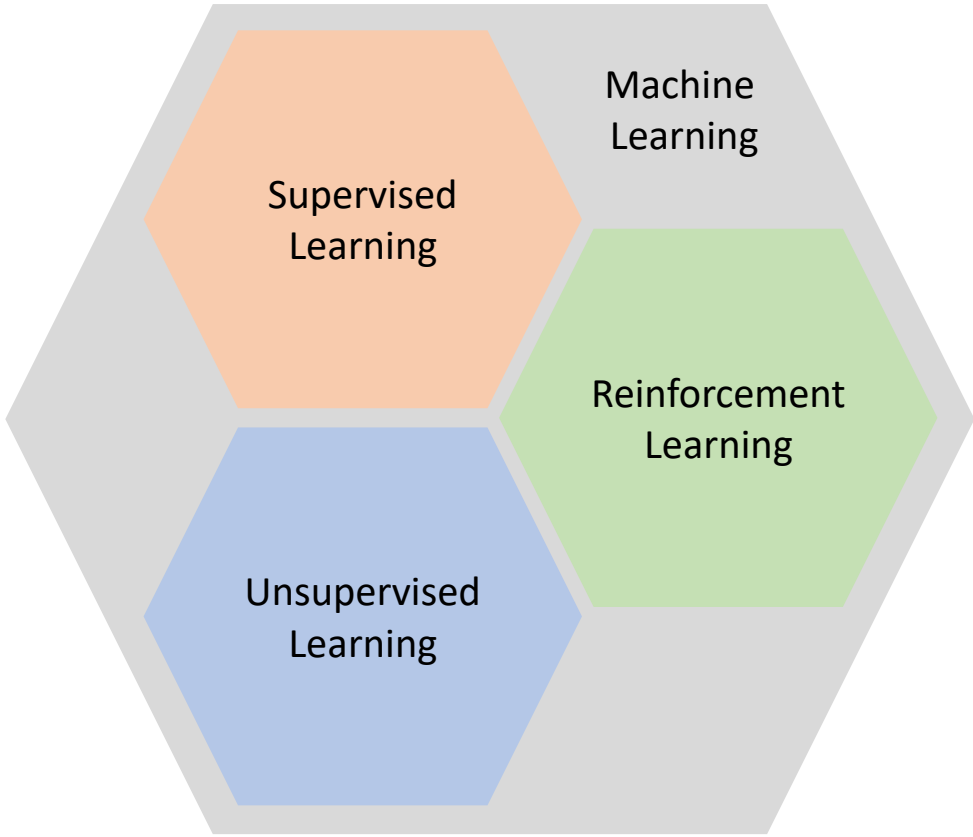
RL in the context of ML:

RL vs Supervised Learning:

- No need for labelled input/output pairs.
- No need to correct suboptimal actions.
- Real-time tasks vs classification and regression.

RL vs Unsupervised Learning:

- Need for feedback of the actions.
- Real-time tasks vs clustering and data reduction.



REINFORCEMENT LEARNING

Reinforcement Learning

Exploration vs Exploitation:

Exploration

- The agent needs to discover its environment.
- Searches to know the consequences of the actions: transitions and rewards.
- It is performed by taking random actions.
- More relevant at the beginning of the learning process.

Exploitation

- The agent needs to perform the best actions.
- Searches to maximize the sum of rewards.
- It is performed by taking the best among the known actions.
- More relevant at the end of the learning process.

REINFORCEMENT LEARNING

Bioinspiration

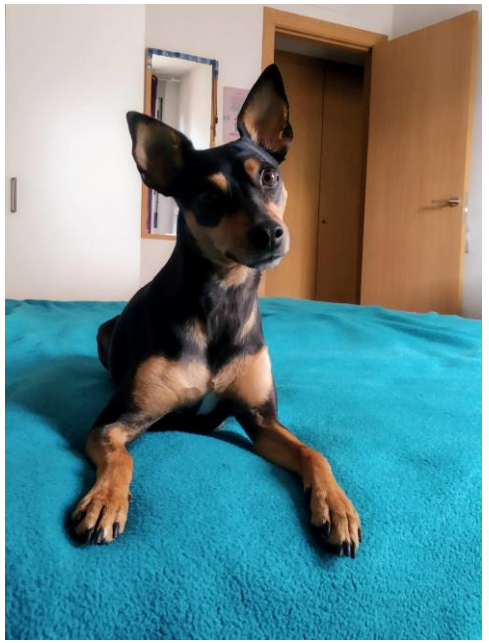
Biological systems must find nutrients, avoid harm and reproduce.

Their environments are dynamic with different types of changes: slow-persistent (e.g., seasonal), sudden-ephemeral (e.g., appearance of predators), and fast-persistent (e.g., destruction of habitat).

For this purpose, they must continuously adapt and learn on multiple timescales.

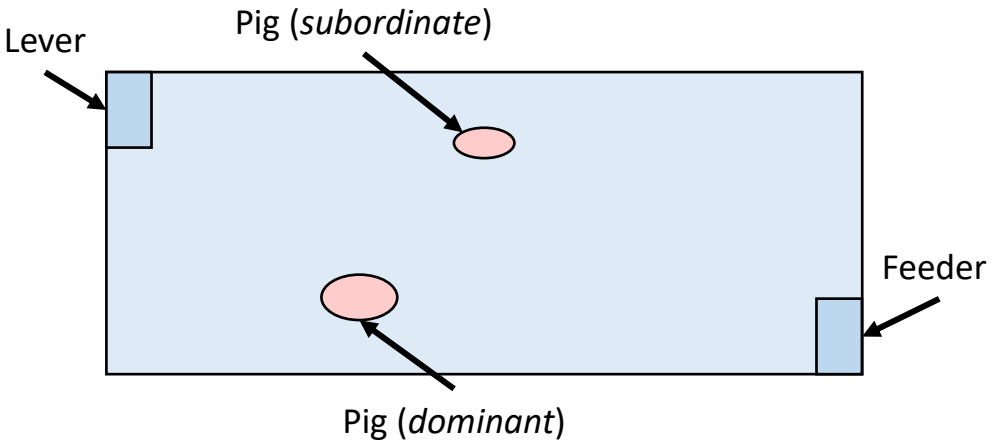
Reinforcement learning and, in some species, imitation learning are relevant.

However, biological and artificial systems learn differently. Biological systems often learn rapidly because the associations between choices and rewards are immediate (e.g., which food to eat). However, artificial systems are usually trained with large amounts of data through statistical optimization.



REINFORCEMENT LEARNING

Bioinspiration



Game theoretical analysis:

		Dominant Pig: Press Lever	Dominant Pig: Don't Press
Subordinate Pig	Press Lever	(2, 6)	(-1, 10)
	Don't Press	**(8, 1)**	(0, 0)

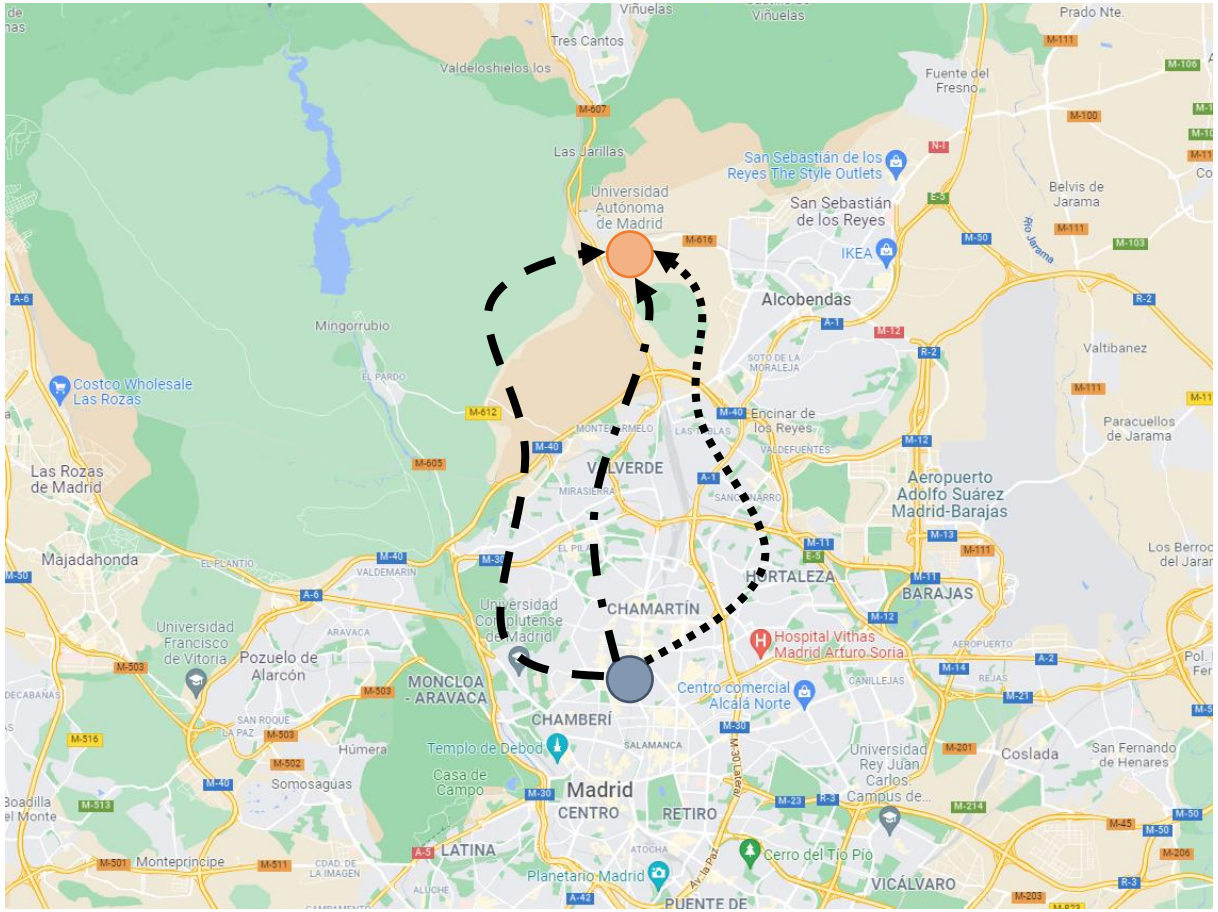
Experimental results:

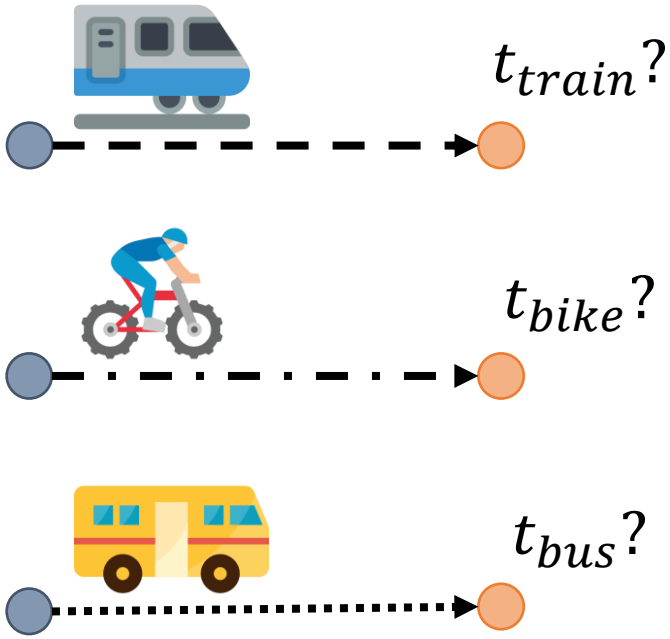
- The pigs are alone:
  - Dominant: 26 times / 5 minutes
  - Subordinate: 16 times / 5 minutes
- The pigs are together:
  - Dominant: 22 times / 5 minutes
  - Subordinate: 1 time / 5 minutes

Baldwin and Meese, **Social behaviour in pigs studied by means of operant conditioning**, *Animal Behavior, 1979.*

REINFORCEMENT LEARNING

Multi-Armed Bandits



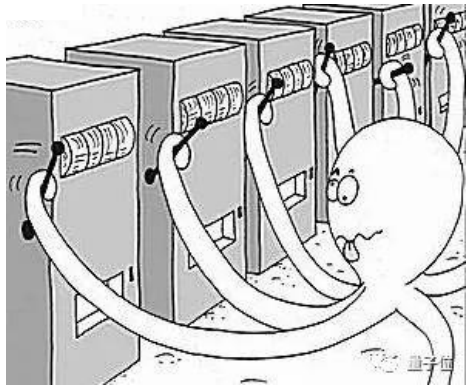


REINFORCEMENT LEARNING

Multi-Armed Bandits

Given a set of K slot machines with unknown reward probability distributions, the objective is to find the sequence of pulls that maximizes the sum of rewards.

For this purpose, the strategy should reach a compromise among exploration (to discover the expected payoff of each machine) and exploitation (to take advantage of the machine with the highest payoff).



Sequence:

1. Choose a bandit (exploration or exploitation).
2. Pull the bandit.
3. Get a reward (prize or nothing).
4. Update your belief.

REINFORCEMENT LEARNING

Multi-Armed Bandits

Exploration

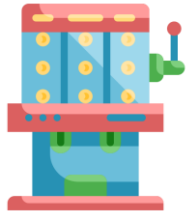
Exploitation

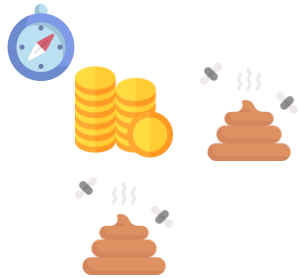
Real success rate

Current success rates

Bandit 1

25%





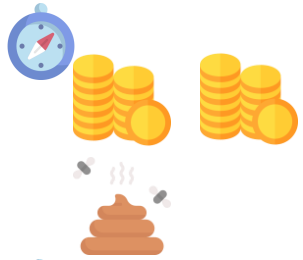
33%

40%

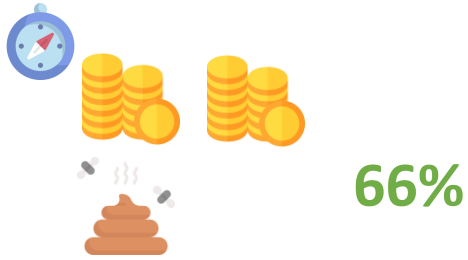
25%

Bandit 2

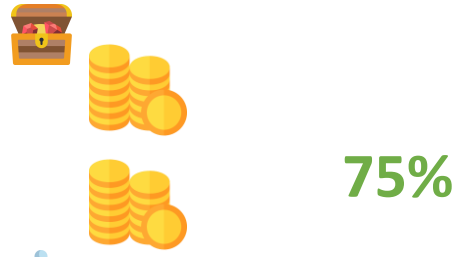
75%



66%



66%

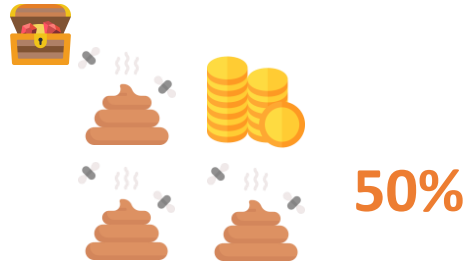


75%

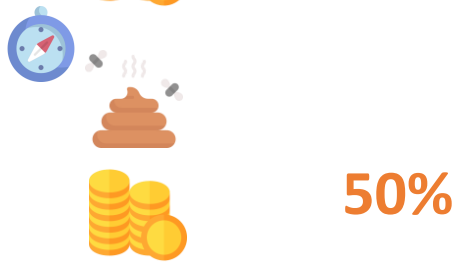
Bandit 3

50%

100%



50%



50%

REINFORCEMENT LEARNING

Multi-Armed Bandits

Types of actions:

- Greedy action: Take the action with the highest expected value (exploitation).
- Non-greedy action: Take any other action (exploration).

Action-value estimation: $Q_t(a) = \frac{\sum_{i=1}^{t-1} 1_{A_i=a} R_i}{\sum_{i=1}^{t-1} 1_{A_i=a}}$ ($1_{A_i=a}$: 1 if action a was taken at i, R_i: reward obtained at i)

First approach: **greedy algorithm**

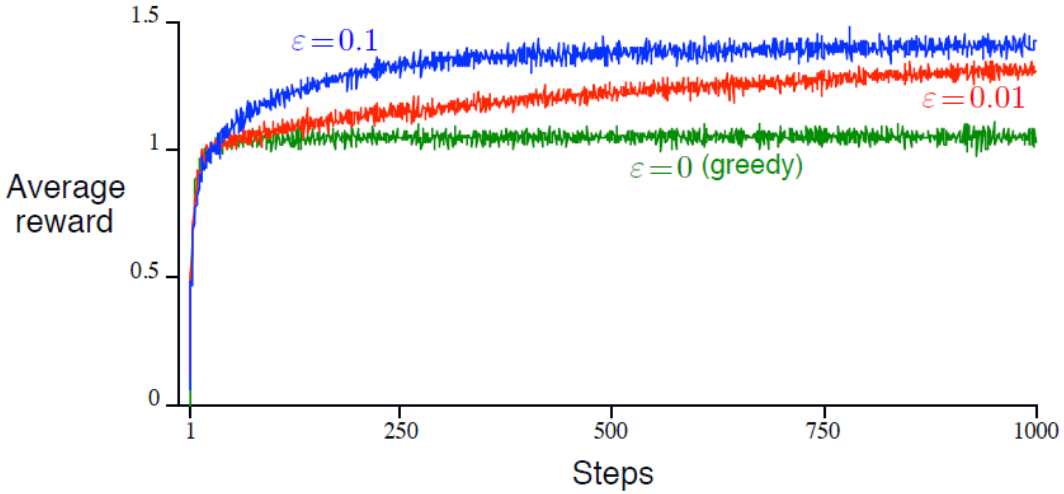
1. Initialize all action-values to 1 (optimistic estimation).
2. Loop (repeat for T steps):
   1. Take the greedy action: $A_t = \underset{a}{\operatorname{argmax}}\, Q_t(a)$.
   2. Update the action-value estimations $Q_t(a)$.

It may not converge to the optimal action.

Multi-Armed Bandits

Second approach: **ε-greedy algorithm**

1. Initialize all action-values to 1 (optimistic estimation).
2. Loop (repeat for T steps):
    1. Generate a random number n.
    2. If $n > \varepsilon$, take the greedy action.
    3. If $n \leq \varepsilon$, take a random action.
    4. Update the action-value estimations $Q_t(a)$.



It continues taking non-greedy actions when the action-value estimations are accurate.

Multi-Armed Bandits

Third approach: **ε-greedy algorithm with decay**

1. Initialize all action-values to 1 (optimistic estimation).
2. Initialize ε with the desired initial value.
3. Loop (repeat for T steps):
    1. Generate a random number n.
    2. If $n > \varepsilon$, take the greedy action.
    3. If $n \leq \varepsilon$, take a random action.
    4. Update the action-value estimations $Q_t(a)$.
    5. Update ε in function of t.

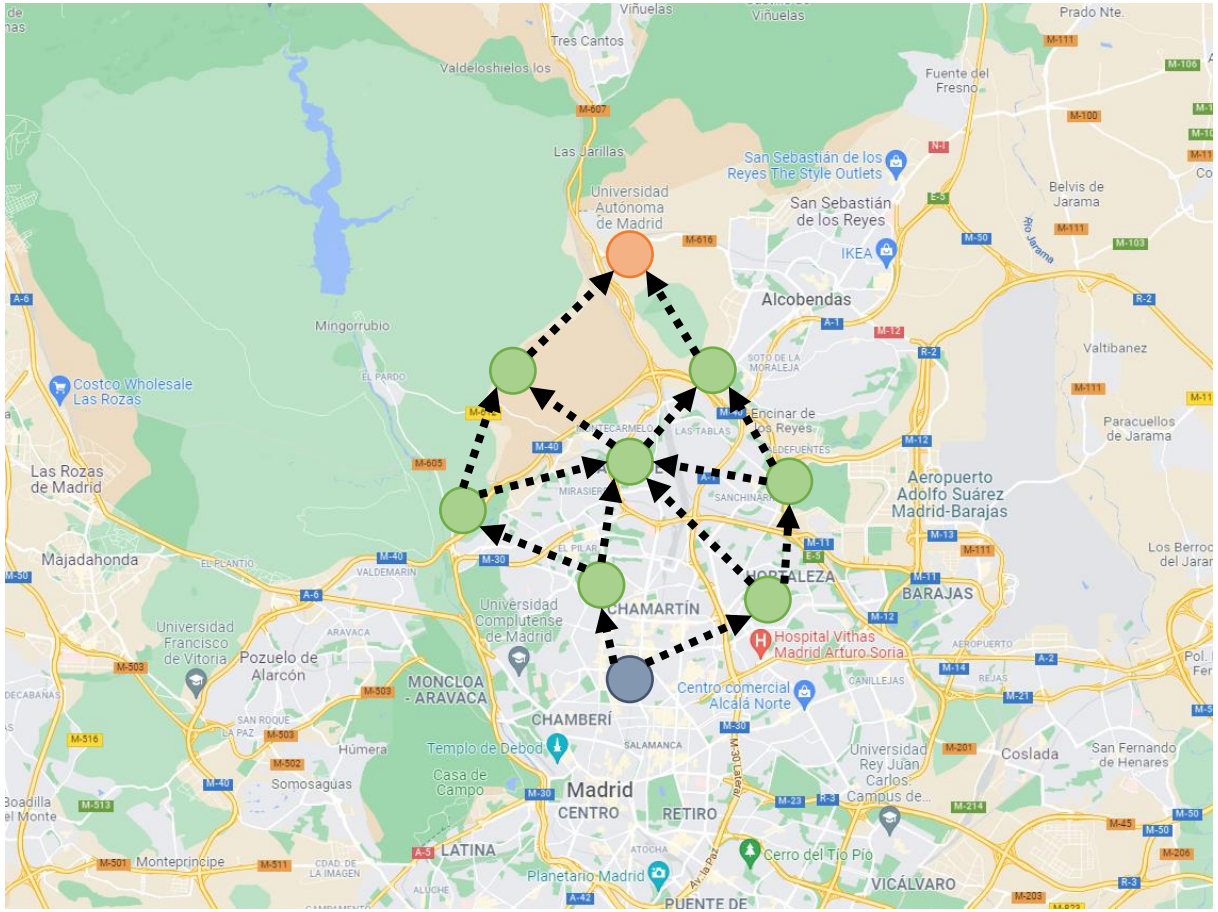
Types of decay for ε:

- Linear: $\varepsilon_t = (1 - \frac{t}{T})\varepsilon_0$
- Exponential: $\varepsilon_t = e^{(1-\frac{t}{T})}\varepsilon_0$
- …

The reward distributions must not change over time.

REINFORCEMENT LEARNING

Markov Decision Processes



Best path?

Markov Decision Processes

Reinforcement learning problems are often represented as Markov Decision Processes (*MDP*):

- **S:** Set of states of the environment that can be observed by the agent.
- **A:** Set of actions that the agent can perform on the environment.
- **P – p(s' : s, a):** Transition matrix, which represents the transition probabilities between s and s' states when action a is performed.
- **R – p(r : s, a):** Reward model, which represents the probability to receive a reward r in the state s when the action a is performed.
- γ: Discount factor, which has a value between 0 and 1 and weights short and long-term rewards.

Markov property: The future state depends on the current state and action and does not depend on the past states and action. **There is no memory.**

Markov Decision Processes

A MDP is solved finding a sequence of actions that maximize its **return**.

Return: Sum of all the rewards weighted with the discount factor.

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$$

Discount

- Low ($\gamma \sim 0$): Short-term thinking. Prioritize current rewards to future ones.
- High ($\gamma \sim 1$): Long-term thinking. Give similar values to current and future rewards.

The discount allows to represent the uncertainty of long-term rewards and avoid the generation of infinite rewards in cyclical processes.

Markov Decision Processes

Bellman Equation:

Value function: The value of a state s is the expected return from this state to a terminal one of the MDP.

$$v(s) = E[G_t|S_t = s]$$

Bellman Equation: The value of a state $v(s)$ can be computed from the immediate reward (R_t) and the discounted value of the next state $v(S_{t+1})$.

$$\begin{gathered} v(s) = E[G_t|S_t = s] \\ = E[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | S_t = s] \\ = \mathrm{E}[R_t + \gamma(R_{t+2} + \gamma^2 R_{t+3} + \ldots)|S_t = s] \\ = \mathrm{E}[R_t + \gamma G_{t+1}|S_t = s] \\ = \mathrm{E}[R_t + \gamma v(S_{t+1})|S_t = s] \end{gathered}$$

$$\boldsymbol{v(s) = \mathrm{E}[R_t + \gamma v(S_{t+1})|S_t = s]}$$

Markov Decision Processes

Policies:

A **policy** is a probability distribution that relates every state with one or multiple actions.

$$\pi(a|s) = P[A_t = a|S_t = s]$$

Features:

- It determines the behavior of the agent in any situation.
- It can be **deterministic** (each state has a unique action) or **stochastic** (each state has a set of actions with different probabilities).
- It depends on the current state and not on the history of past states.

Markov Decision Processes

Value functions:

State-value function: Expected return from the state s following the policy π.

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

⇩ Bellman Equation

$$v_\pi(s) = E_\pi[R_t + \gamma v_\pi(S_{t+1})|S_t = s]$$

Action-value function: Expected return from the state s, first taking the action a and then following the policy π.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = t]$$

⇩ Bellman Equation

$$q_\pi(s,a) = E_\pi[R_t + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Markov Decision Processes

Value functions:

Optimal state-value function: Maximum return that can be obtained from the state s considering all the policies.

$$v^*(s) = \max_{\pi} v_\pi(s)$$

Optimal action-value function: Maximum return that can be obtained from the state s, first taking the action a and then considering all the policies.

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

Bellman Optimality Equation: The maximum return from a state is obtained by choosing the action with the maximum value in that state.

$$v^*(s) = \max_{a} q^*(s,a)$$

Markov Decision Processes

Methods:

Value-Based Algorithms: They compute the state-value and action-value functions and search the actions that provide the highest return.

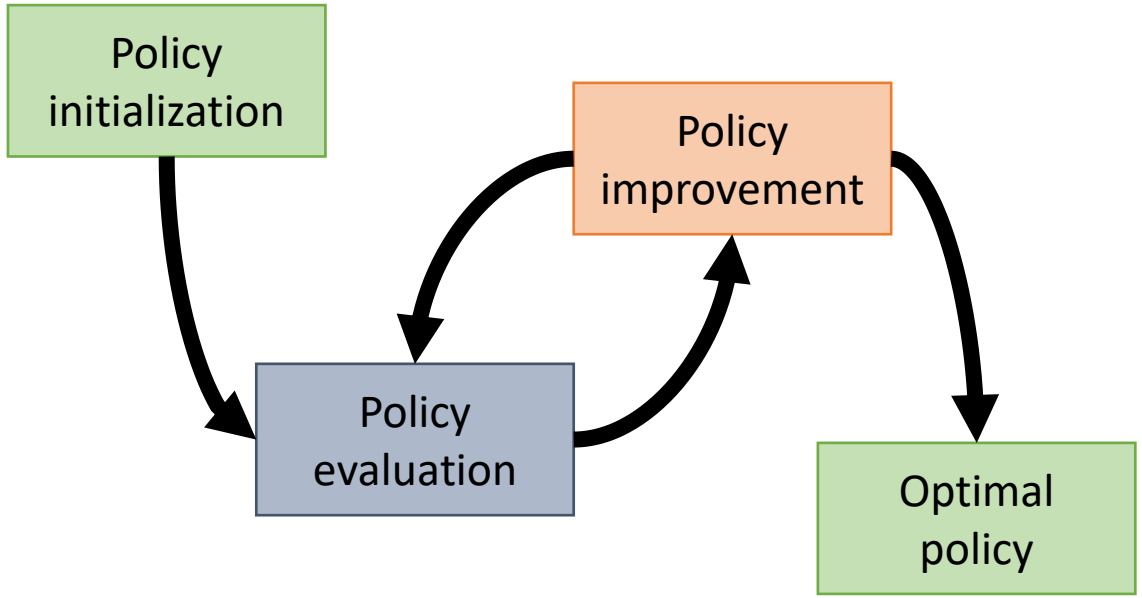
Policy-Based Algorithms: They explicitly build a policy, which relates each state to one or multiple actions, and then perform an optimization of this policy.

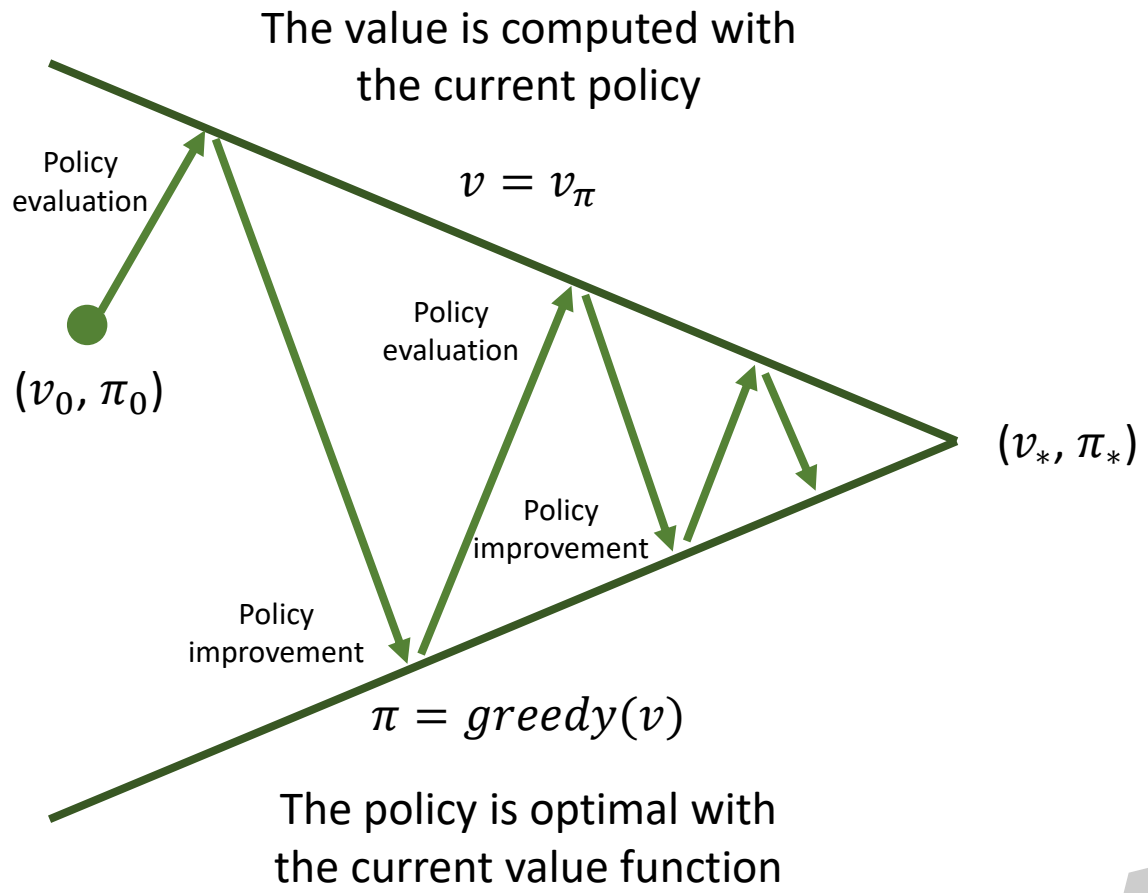
Model-Based Algorithms: They create a model of the environment that represents state transitions and rewards. The model allows the agent to plan the actions before executing them.

Actor-Critic Algorithms: They combine value-based and policy-based algorithms, creating a critic, which estimates the value functions, and an actor, which updates the policy.

Policy-Based Algorithms

Policy iteration:





Policy-Based Algorithms

Policy evaluation: Computation of the state-value function for a policy $v_\pi(s)$.

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_t + \gamma v_\pi(S_{t+1})|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

$\pi(a|s)$: Probability for taking the action a in the state s according to the policy π.
$p(s', r|s, a)$: Probability for changing from state s to s' with a reward r when taking the action a.

Policy-Based Algorithms

Policy evaluation:

1) Initialize v_0 arbitrarily for each state s except for the terminals that must be 0.
2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

3) Repeat until the algorithm converges, i.e., the difference between v_{k+1} and v_k is neglectable.

$$\max_{s \in S} |v_{k+1}(s) - v_k(s)| < \theta$$

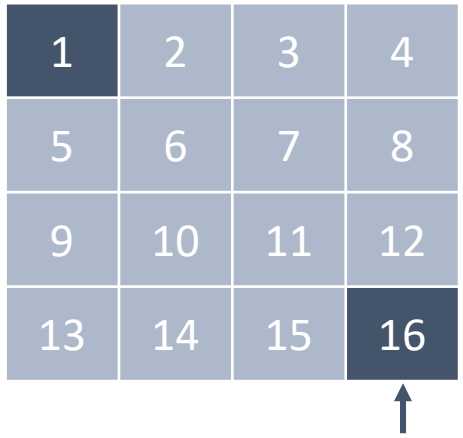
Policy-Based Algorithms

Policy evaluation:

GridWorld

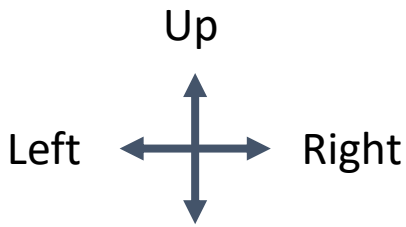
States

Initial state



Final state

Policy

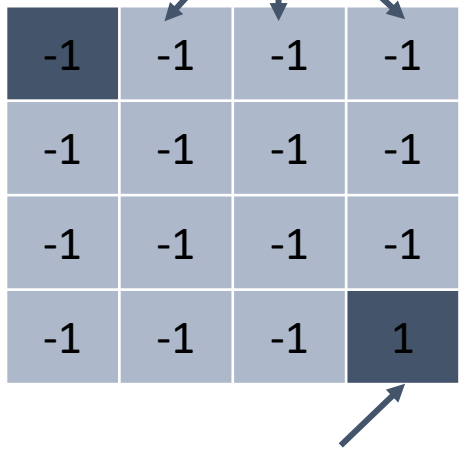


Discount

$\gamma = 1$

Rewards

Penalties to perform many moves



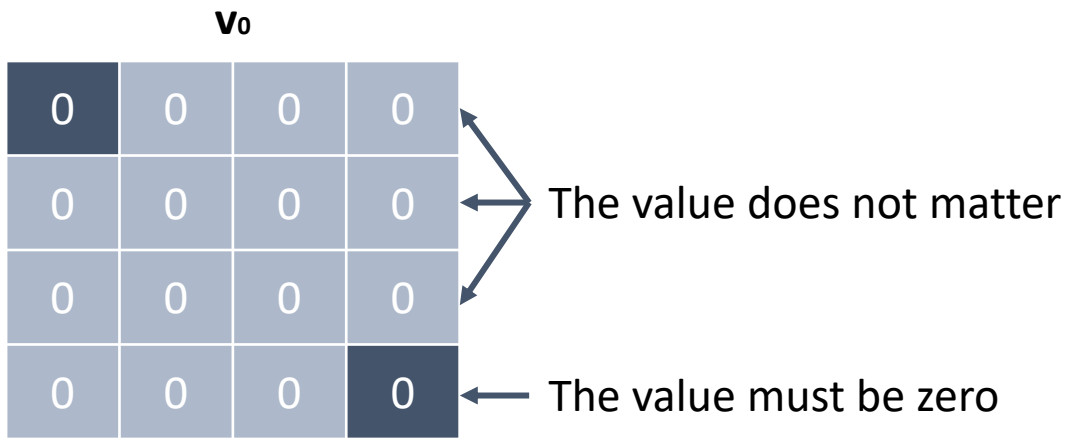
Prize to get the goal

Policy-Based Algorithms

Policy evaluation:

GridWorld

1) Initialize v_0 arbitrarily for each state s except for the terminals that must be 0.



Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

v_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

-1			

$1 \leftarrow 1 \quad 1 \rightarrow 2 \quad 1 \uparrow 1 \quad 1 \downarrow 5$

$$v_1(1) = \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(-1) = -1$$

Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

v_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

-1	-1	-1	-1
-1	-1	-1	

7 ← 6 7 → 8 7 ↑ 3 7 ↓ 11

$$v_1(7) = \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(-1) = -1$$

Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

v_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-0.5

$12 \leftarrow 11 \quad 12 \rightarrow 12 \quad 12 \uparrow 8 \quad 12 \downarrow 16$

$$v_1(12) = \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(-1) + \frac{1}{4}(1) = -0.5$$

Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

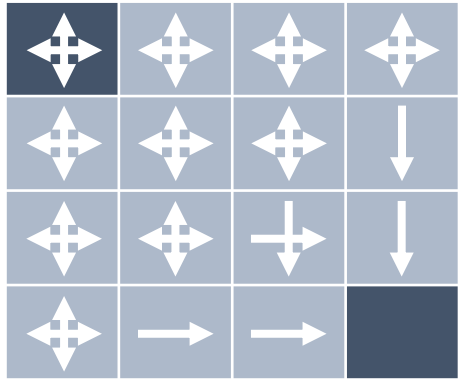
v_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

v_1

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-0.5
-1	-1	-0.5	0

Best actions



Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

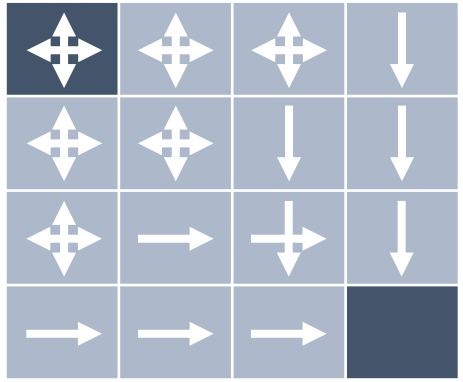
v_1

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-0.5
-1	-1	-0.5	0

v_2

-2	-2	-2	-2
-2	-2	-2	-1.88
-2	-2	-1.75	-1.13
-2	-1.88	-1.13	0

Best actions



$$v_2(3) = \frac{1}{4}(-1-1) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-1) = -2$$

$$v_2(8) = \frac{1}{4}(-1-1) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-0.5) = -1.875$$

$$v_2(11) = \frac{1}{4}(-1-1) + \frac{1}{4}(-1-0.5) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-0.5) = -1.75$$

$$v_2(15) = \frac{1}{4}(-1-1) + \frac{1}{4}(1+0) + \frac{1}{4}(-1-1) + \frac{1}{4}(-1-0.5) = -1.125$$

Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

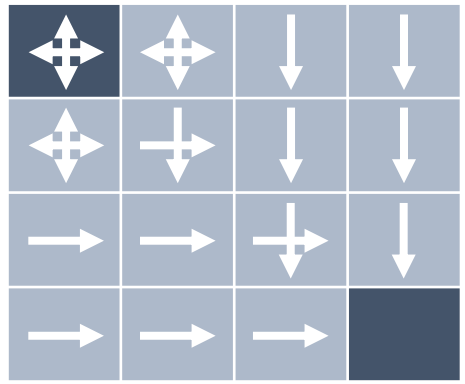
v_2

-2	-2	-2	-2
-2	-2	-2	-1.88
-2	-2	-1.75	-1.13
-2	-1.88	-1.13	0

v_3

-3	-3	-3	-2.97
-3	-3	-2.9	-2.75
-3	-2.9	-2.57	-1.69
-2.97	-2.75	-1.69	0

Best actions



Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

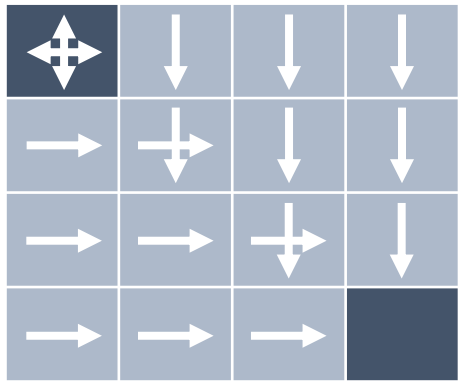
v_3

-3	-3	-3	-2.97
-3	-3	-2.9	-2.75
-3	-2.9	-2.57	-1.69
-2.97	-2.75	-1.69	0

v_4

-4	-4	-3.96	-3.92
-4	-3.95	-3.83	-3.58
-3.96	-3.83	-3.3	-2.25
-3.92	-3.58	-2.25	0

Best actions



Policy-Based Algorithms

Policy evaluation:

GridWorld

2) Update v_{k+1} for each state s from v_k for the successor states s'.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

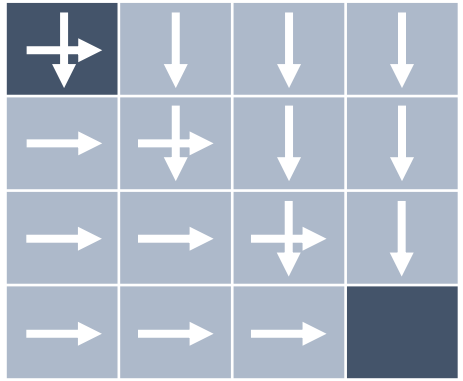
v_4

-4	-4	-3.96	-3.92
-4	-3.95	-3.83	-3.58
-3.96	-3.83	-3.3	-2.25
-3.92	-3.58	-2.25	0

v_5

-5	-4.98	-4.92	-4.85
-4.98	-4.91	-4.7	-4.39
-4.92	-4.7	-4.04	-2.78
-4.85	-4.39	-2.78	0

Best actions



We are evaluating the four directions random policy, so these best actions are not the optimal policy.

Policy-Based Algorithms

Policy improvement: Change actions in the policy to get more value.

Sequence:

1) We know a policy π with a value function v_π.
2) We discover an action **a** that provides more value than the policy.

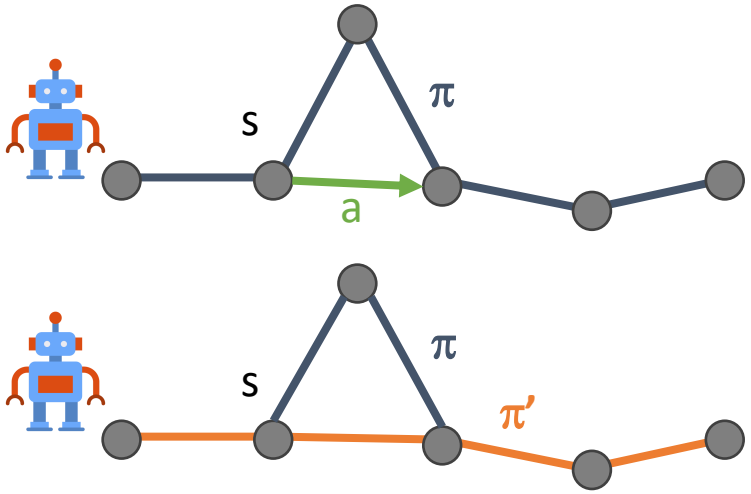
$$q_\pi(s,a) \geq v_\pi(s)$$

3) We can design a new policy π' from π adding **a**.

$$\pi' \equiv \begin{cases} a, & S = s \\ \pi, & \forall S \neq s \end{cases}$$

4) The new policy π' provides more value tan the previous one π.

$$v_{\pi'}(s) \geq v_\pi(s)$$



Policy-Based Algorithms

Policy iteration:

1) Initialization:
Initialize the policy $\pi(s) \in A(s)$ and the value function $\mathrm{V}(s) \in R$ arbitrarily for every state $s \in S$.

2) Policy evaluation:
Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$\mathrm{v} \leftarrow \mathrm{V}(s)$

$\mathrm{V}(\mathrm{s}) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

3) Policy improvement:
Loop (repeat for every state $s \in S$):

$\pi_0 \leftarrow \pi$

$\pi \leftarrow \underset{a}{\mathrm{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $\pi_0 = \pi$, return $v_* \approx V$ y $\pi_* \approx \pi$. Otherwise, go back to 2).

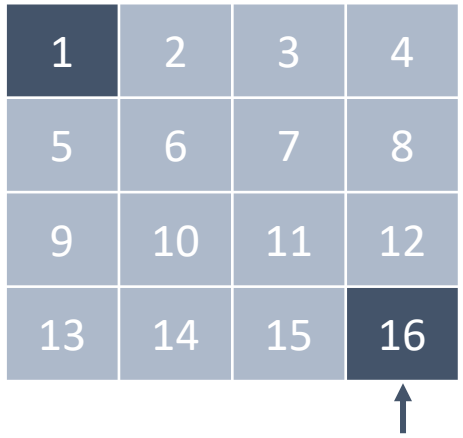
Policy-Based Algorithms

Policy iteration:

GridWorld

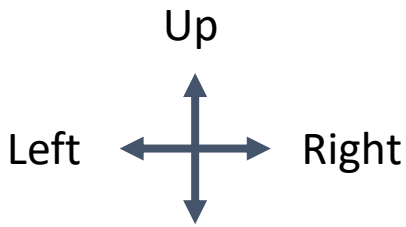
States

Initial state



Final state

Possible actions

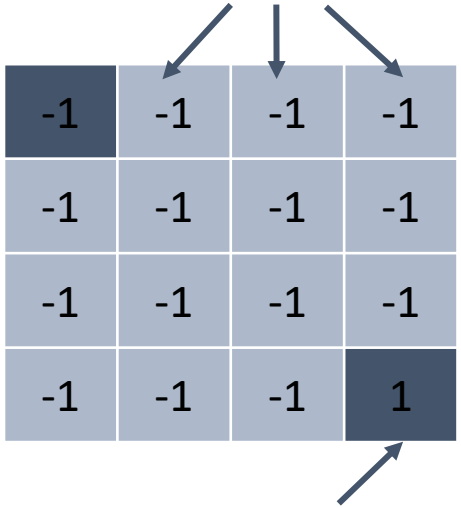


Discount

$\gamma = 1$

Rewards

Penalties to perform many moves



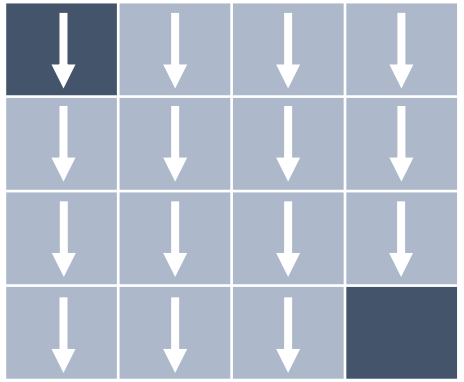
Prize to get the goal

Policy-Based Algorithms

Policy iteration:

1) Initialization:
Initialize the policy $\pi(s) \in A(s)$ and the value function $\mathrm{V}(s) \in R$ arbitrarily for every state $s \in S$.

π(s)



V(s)

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

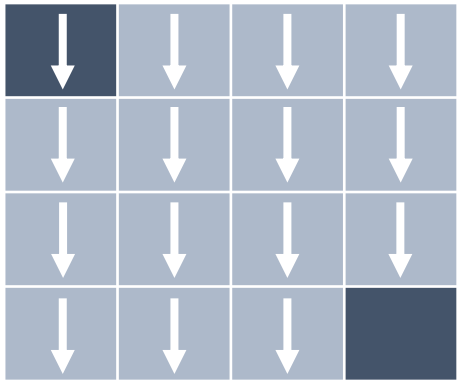
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

π(s)



V(s)

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	1
-1	-1	-1	1

Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

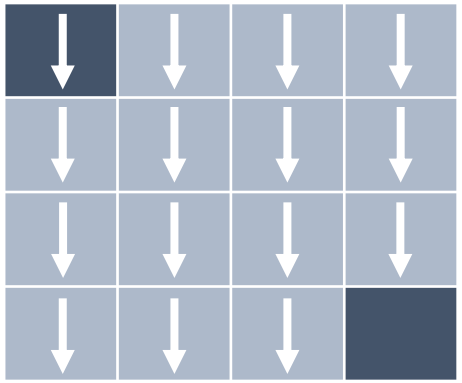
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	1
-1	-1	-1	1

π(s)



V(s)

-2	-2	-2	-2
-2	-2	-2	0
-2	-2	-2	2
-2	-2	-2	2

Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

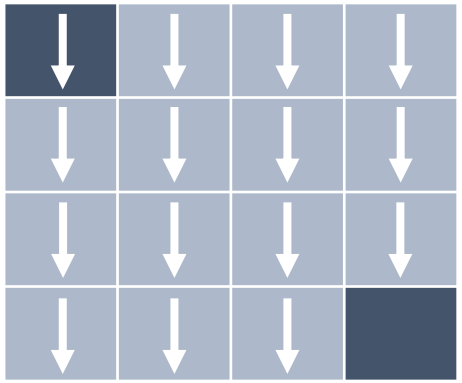
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-2	-2	-2	-2
-2	-2	-2	0
-2	-2	-2	2
-2	-2	-2	2

π(s)



V(s)

-3	-3	-3	-1
-3	-3	-3	1
-3	-3	-3	3
-3	-3	-3	3

Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

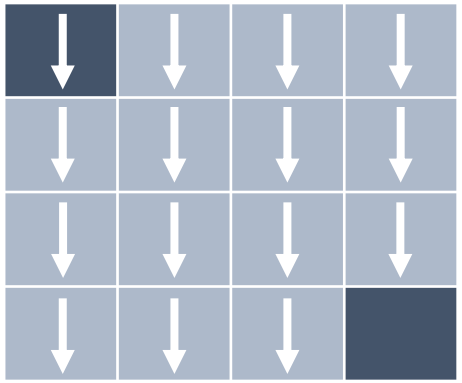
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-3	-3	-3	-1
-3	-3	-3	1
-3	-3	-3	3
-3	-3	-3	3

π(s)



V(s)

-4	-4	-4	0
-4	-4	-4	2
-4	-4	-4	4
-4	-4	-4	4

Policy-Based Algorithms

Policy iteration:

3) Policy improvement:

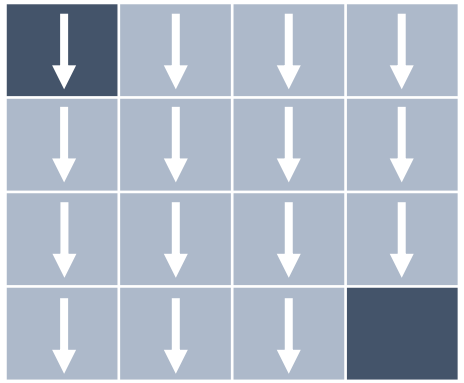
Loop (repeat for every state $s \in S$):

$\pi_0 \leftarrow \pi$

$\pi \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $\pi_0 = \pi$, return $v_* \approx V$ y $\pi_* \approx \pi$. Otherwise, go back to 2).

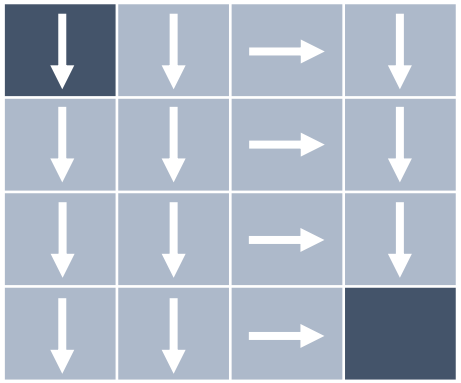
π_0



V(s)

-4	-4	-4	0
-4	-4	-4	2
-4	-4	-4	4
-4	-4	-4	4

π



Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

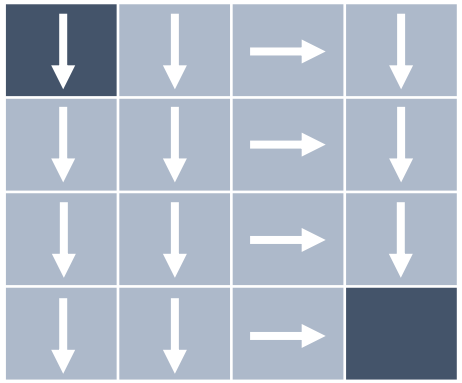
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-4	-4	-4	0
-4	-4	-4	2
-4	-4	-4	4
-4	-4	-4	4

π(s)



V(s)

-5	-5	-1	1
-5	-5	1	3
-5	-5	3	5
-5	-5	5	5

Policy-Based Algorithms

Policy iteration:

3) Policy improvement:

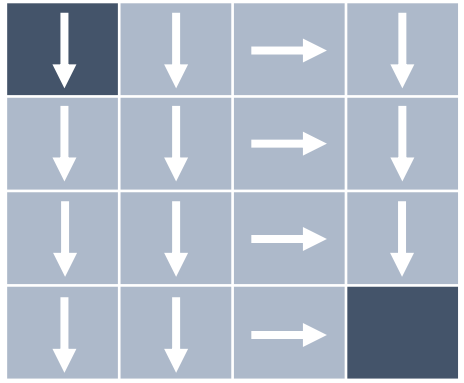
Loop (repeat for every state $s \in S$):

$\pi_0 \leftarrow \pi$

$\pi \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $\pi_0 = \pi$, return $v_* \approx V$ y $\pi_* \approx \pi$. Otherwise, go back to 2).

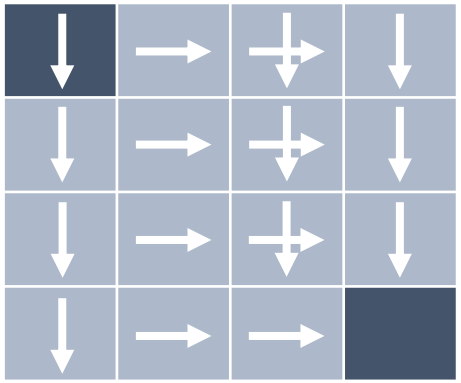
π_0



V(s)

-5	-5	-1	1
-5	-5	1	3
-5	-5	3	5
-5	-5	5	5

π



Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

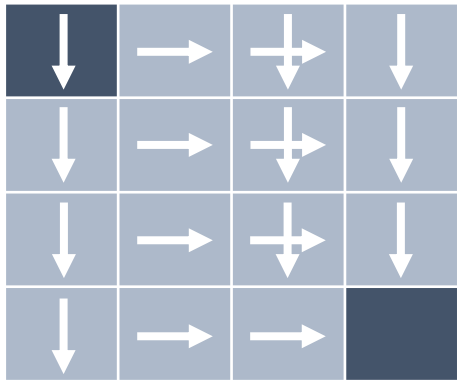
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-5	-5	-1	1
-5	-5	1	3
-5	-5	3	5
-5	-5	5	5

π(s)



V(s)

-6	-2	0	2
-6	0	2	4
-6	2	4	6
-6	4	6	6

Policy-Based Algorithms

Policy iteration:

3) Policy improvement:

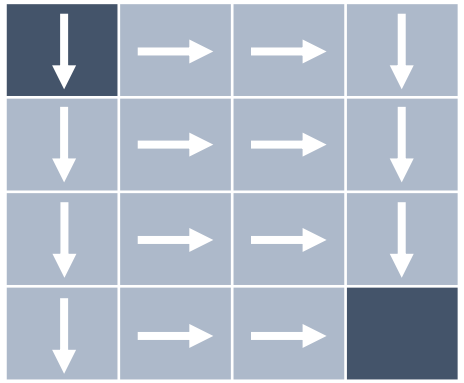
Loop (repeat for every state $s \in S$):

$\pi_0 \leftarrow \pi$

$\pi \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $\pi_0 = \pi$, return $v_* \approx V$ y $\pi_* \approx \pi$. Otherwise, go back to 2).

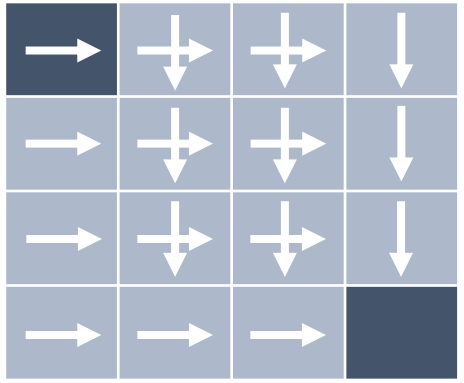
π_0



V(s)

-6	-2	0	2
-6	0	2	4
-6	2	4	6
-6	4	6	6

π



Policy-Based Algorithms

Policy iteration:

2) Policy evaluation:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

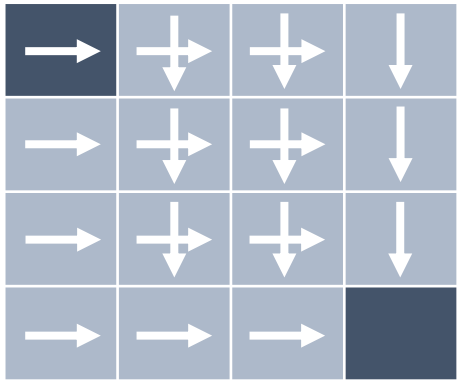
$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

v

-6	-2	0	2
-6	0	2	4
-6	2	4	6
-6	4	6	6

π(s)



V(s)

-3	-1	1	3
-1	1	3	5
1	3	5	7
3	5	7	7

Policy-Based Algorithms

Policy iteration:

3) Policy improvement:

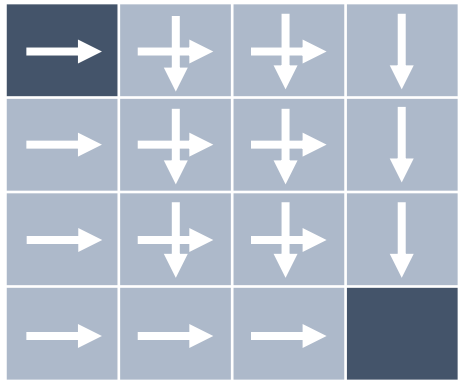
Loop (repeat for every state $s \in S$):

$\pi_0 \leftarrow \pi$

$\pi \leftarrow \underset{a}{\text{argmax}} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $\pi_0 = \pi$, return $v_* \approx V$ y $\pi_* \approx \pi$. Otherwise, go back to 2).

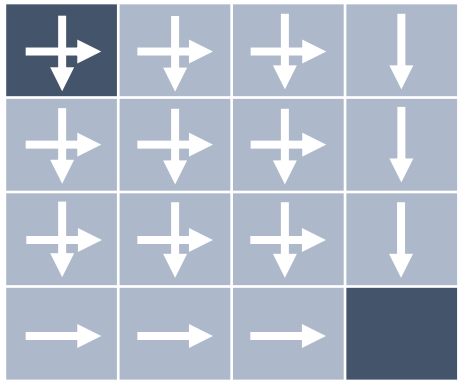
π_0



V(s)

-3	-1	1	3
-1	1	3	5
1	3	5	7
3	5	7	7

π



Policy-Based Algorithms

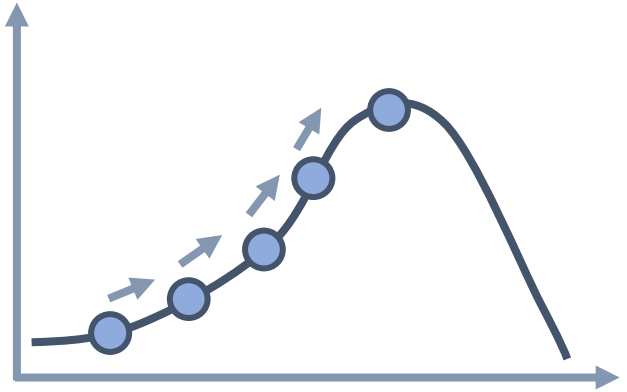
Policy gradient:

Use a parametrized policy instead of the value function:

$\pi(a|s,\theta) = P\{A_t = a|S_t = s, \theta_t = \theta\}$ θ: Vector of the policy parameters

Optimize this policy through gradient descent methods:

$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta)}$ $\widehat{\nabla J(\theta)}$: Estimator of the performance gradient



The policy can be parametrized in any way as long as $\pi(a|s,\theta)$ is differentiable with respect to the parameters θ, i.e., its partial derivatives with respect to them $\nabla\pi(a|s,\theta)$ exist and are finite for every state $s \in S$, action $a \in A$, and parameter $\theta \in R^{d'}$.

Additionally, the policy must be non-deterministic to ensure that it explores the environment and does not converge in local maxima: $\pi(a|s,\theta) \in (0,1), \forall s, a, \theta$

Policy-Based Algorithms

Policy gradient:

Parametrization:

1) A preference function is assigned to each state-action pair to define its probability:

$$h(s, a, \theta) \in R$$

2) A soft-max distribution is applied to ensure that the sum of probabilities is one:

$$\pi(a|s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,a,\theta)}}$$

Note: $h(s, a, \theta)$ can take multiple expressions…

- Linear: $h(s, a, \theta) = \theta^T x(s, a)$.
- Neural network (θ are the weights).
- …

Policy-Based Algorithms

Policy gradient:

Policy gradient theorem:

It allows to estimate the performance gradient J with respect to the parameter θ without computing derivatives.

$$\nabla J(\theta) \ \alpha \ \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\theta)$$

α: Proportionality: the constant is 1 for continuous tasks and the episode length for episodic tasks.
$\mu(s)$: State distribution: fraction of time spent in a state over the total.
$q_\pi(s,a)$: Action-value function.
$\nabla \pi(a|s,\theta)$: Policy gradient with respect to the parameter θ.

Policy-Based Algorithms

REINFORCE:

Input: Parametrized policy $\pi(a|s,\theta)$.
Parameters: Episode length α.

Initialize the parameters θ with random values.
Loop (repeat for each episode until convergence):
Generate an episode $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$ following the policy $\pi(a|s,\theta)$.
Loop (Repeat for each step of the episode $t = 0, 1, \dots, T-1$):
$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
$\theta = \theta + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\theta)$

Note: $\nabla \ln x = \frac{\nabla x}{x}$

For the deduction of the update rule see: Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Value-Based Algorithms

One disadvantage of the policy iteration is that each iteration implies evaluating one policy, which requires sweeping multiple times the table of states.

Policy evaluation:

Loop (repeat until $\Delta < \theta$): ← Repetition until the values converge.

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

} Full sweep of the table of states.

An alternative is to sweep once the table of states in every policy evaluation, which gives place to **value iteration**.

Value-Based Algorithms

Value iteration:

1) Initialization:

Initialize the value function $\mathrm{V}(s) \in R, \forall s \in S$ arbitrarily except for the terminals that must be 0.

2) Value iteration:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$\mathrm{v} \leftarrow \mathrm{V}(s)$

$\mathrm{V}(\mathrm{s}) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

Return: $\pi_* \approx \pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$.

REINFORCEMENT LEARNING

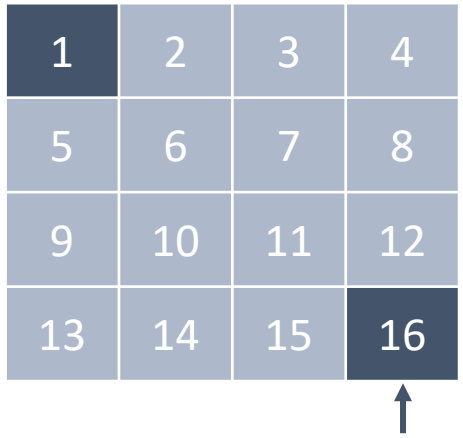
Value-Based Algorithms

Value iteration:

GridWorld

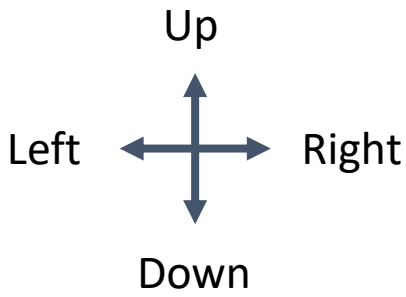
States

Initial state



Final state

Possible actions

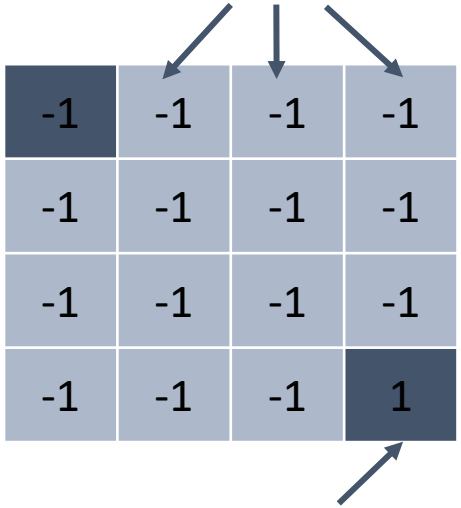


Discount

$\gamma = 1$

Rewards

Penalties to perform many moves



Prize to get the goal

Value-Based Algorithms

Value iteration:

1) Initialization:

Initialize the value function $V(s) \in R, \forall s \in S$ arbitrarily except for the terminals that must be 0.

V(s)

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Value-Based Algorithms

Value iteration:

2) Value iteration:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

Return: $\pi_* \approx \pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$.

V_0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	1
-1	-1	1	1

V_2

-2	-2	-2	-2
-2	-2	-2	0
-2	-2	0	2
-2	0	2	2

V_3

-3	-3	-3	-1
-3	-3	-1	1
-3	-1	1	3
-1	1	3	3

Value-Based Algorithms

Value iteration:

2) Value iteration:

Loop (repeat until $\Delta < \theta$):

$\Delta \leftarrow 0$

Loop (repeat for every state $s \in S$):

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

Return: $\pi_* \approx \pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$.

v_3

-3	-3	-3	-1
-3	-3	-1	1
-3	-1	1	3
-1	1	3	3

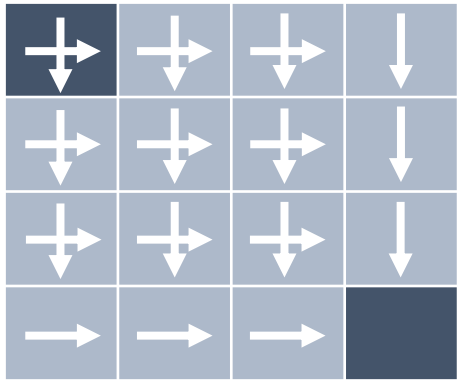
v_4

-4	-4	-2	0
-4	-2	0	2
-2	0	2	4
0	2	4	4

v_5

-5	-3	-1	1
-3	-1	1	3
-1	1	3	5
1	3	5	5

π_*



Value-Based Algorithms

SARSA:

SARSA: State-Action-Reward-State-Action

The value update function depends on the current state (S_t), the selected action in that state (A_t), the reward received when taking that action (R_t), the next state (S_{t+1}), and the action selected in that state (A_{t+1}).

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

$Q(S_t, A_t)$ y $Q(S_{t+1}, A_{t+1})$: Estimation of the value generated in the current and next state when taking the current and next action obtained from the process data.
α: Episode length.
γ: Discount.

Attention: If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is null.

Value-Based Algorithms

SARSA:

Parameters: Episode length α, balance between exploration and exploitation ε.

Initialize the values $Q(s, a)$ arbitrarily for every state $s \in S$ and action $a \in A(s)$ except for the terminal states $Q(terminal, a) = 0$.

Loop (repeat for each episode until convergence):
- Initialize S
- Choose A in S using a policy obtained from Q
- Loop (repeat for each step until S is terminal):
  - Take the action A, observe R y S'
  - Choose A' in S' using a policy obtained from Q
  - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
  - $S \leftarrow S'; A \leftarrow A'$

Value-Based Algorithms

SARSA:

Choose A in S using a policy obtained from Q?

We have estimated the values of all the actions in the state S: Q(S, a).

- We can choose the action that provides the maximum value (**exploitation**):

$$A \leftarrow \underset{a}{\operatorname{argmax}}\, Q(S, a)$$

- Or we can choose other action to estimate its value (**exploration**):

$$A \leftarrow \operatorname{random}(a)$$

In practice, we can combine both options (***ε*-greedy and *ε*-soft methods**):

$$\operatorname{random}(n) < \varepsilon \Rightarrow A \leftarrow random(a)$$
$$\operatorname{random}(n) \geq \varepsilon \Rightarrow A \leftarrow \underset{a}{\operatorname{argmax}}\, Q(S, a)$$

REINFORCEMENT LEARNING

Value-Based Algorithms

Q-Learning:

Q: Q(S,A) – Value of taking the action A in the state S.

The value update functions depends on the current state (S_t), the action selected in that state (A_t), the reward obtained when selecting that action (R_t), the next state (S_{t+1}) and the best action in that state.

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

$Q(S_t, A_t)$: Estimation of the value of taking the current action in the current state.
$\max_a Q(S_{t+1}, a)$: Estimation of the value of taking the best action in the next state.
α: Episode length.
γ: Discount.

Value-Based Algorithms

Q-Learning:

Parameters: Episode length α, balance between exploration and exploitation ε.

Initialize the values $Q(s,a)$ arbitrarily for every state $s \in S$ and action $a \in A(s)$ except for the terminal states $Q(terminal,a) = 0$.

Loop (repeat for each episode until convergence):
- Initialize S
- Loop (repeat for each step until S is terminal):
  - Select A in S using a policy from Q
  - Take the action A, observe R y S'
  - $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  - $S \leftarrow S'$