Fundamentals of (Deep) Reinforcement Learning

INTELLIGENT SYSTEMS FOR PATTERN RECOGNITION (ISPR)

DAVIDE BACCIU – DIPARTIMENTO DI INFORMATICA - UNIVERSITA’ DI PISA

DAVIDE.BACCIU@UNIPI.IT

Lecture Outline

- RL Fundamentals
- Model based RL
- Model free RL
- Hints of deep reinforcement learning

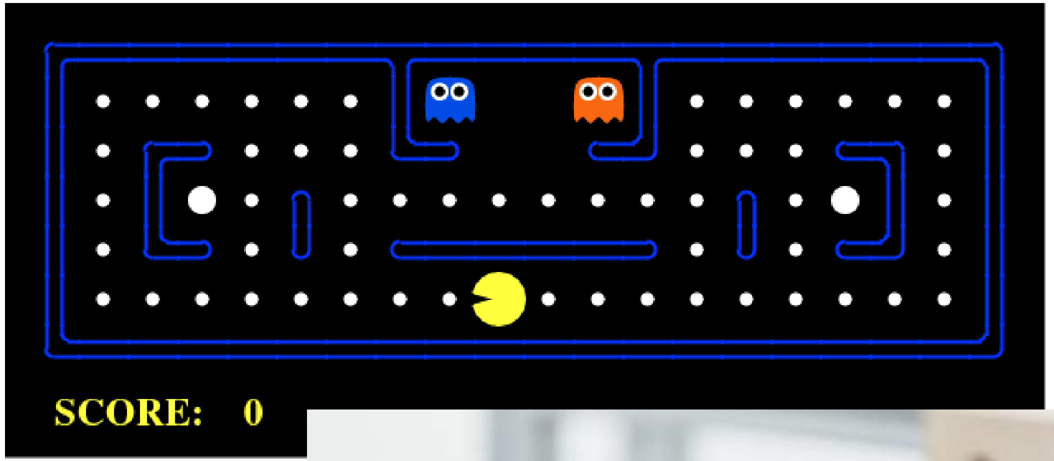
UNIVERSITÀ DI PISA

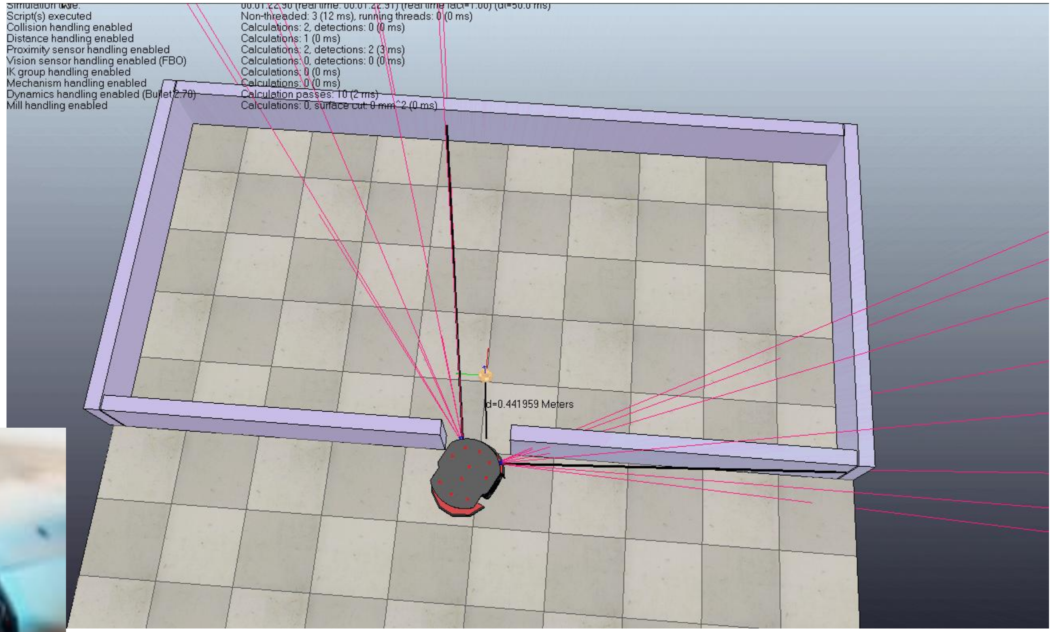
Introduction & Formal Model

What characterizes Reinforcement Learning (vs other ML tasks)?

- No supervisor: only a *reward* signal
- Delayed asynchronous feedback
- Time matters (sequential data, continual learning)
- Agent’s actions affect the subsequent data it receives (inherent non-stationarity)

(Some) RL Tasks



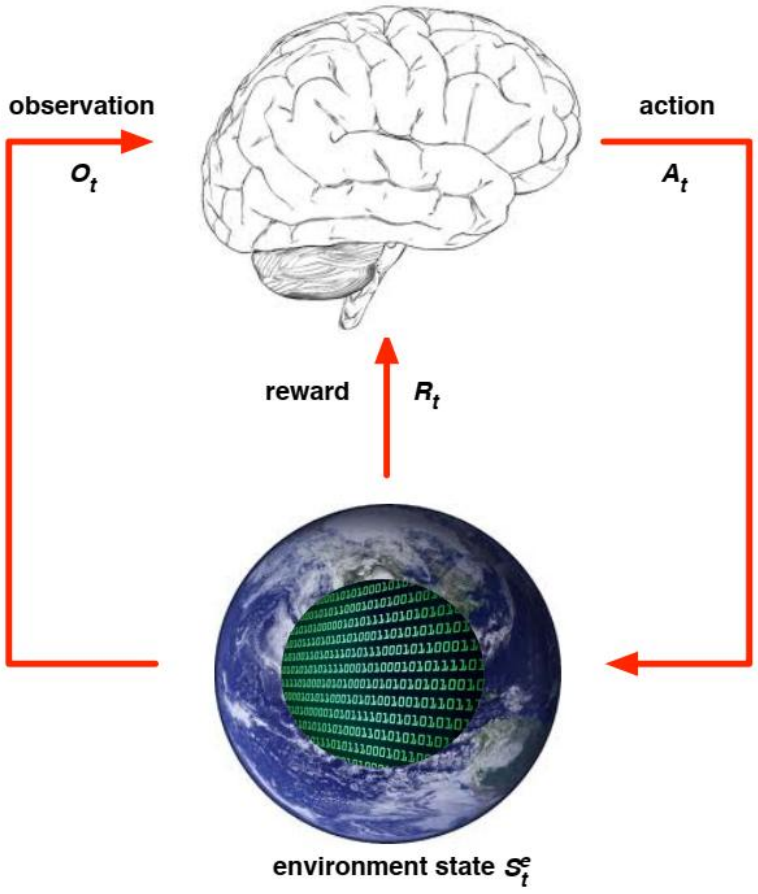


Rewards

- A reward R_t is a scalar feedback signal
- Indicates how well agent is doing at step t
- The agent's job is to maximise cumulative reward
- Reinforcement learning is based on the reward hypothesis
- All goals can be described by the maximisation of expected cumulative reward

Agents and Environments

- S_t^e is the environment e private representation at time t
- S_t^a the internal representation owned by agent a
- Full observability $\Longrightarrow$ Agent directly observes the environment state
$$O_t = S_t^a = S_t^e$$
- Formally this is a Markov Decision Process (MDP)



Markov Decision Process

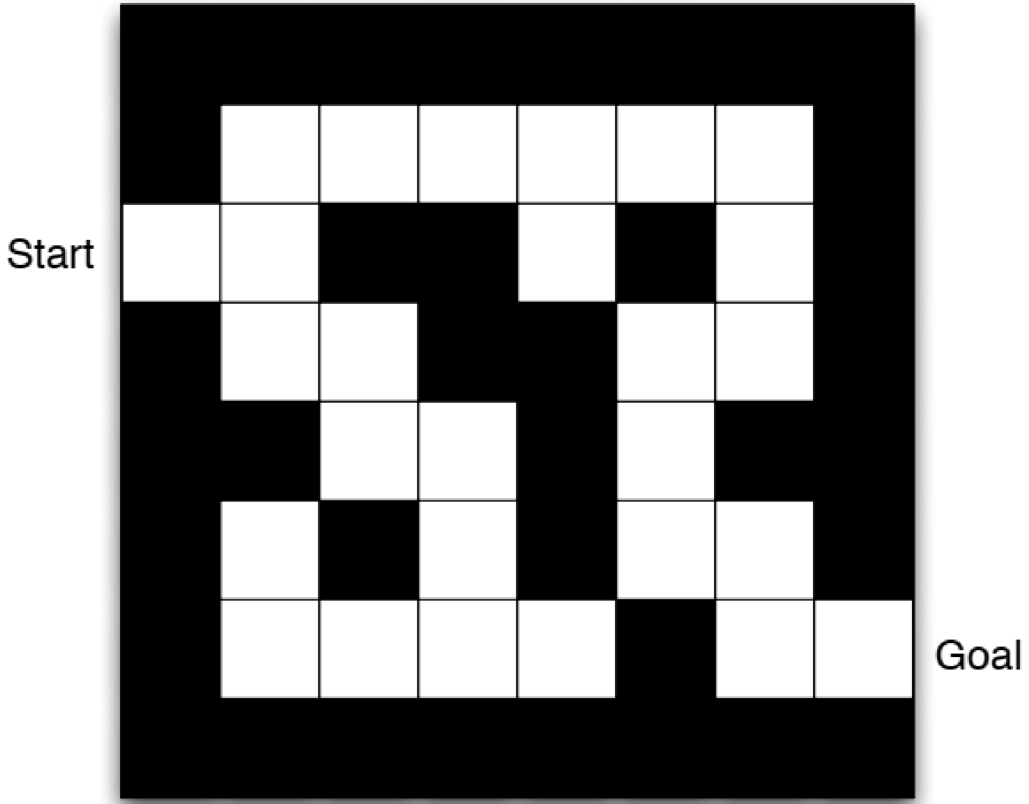
A Markov Decision Process (MDP) is a Markov chain with rewards and actions. It is an environment in which all states are Markov

Definition (Markov Decision Process)

A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \boldsymbol{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions a
- $\boldsymbol{P}$ is a state transition matrix, s.t. $P^a_{ss'} = P(S_{t+1} = s' | S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function, s.t. $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor, $\gamma \in [0,1]$

A Forever Classic - The Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent location

RL Goal - Return Maximization

Definition (Return)

The return G_t is the total discounted reward from time-step t

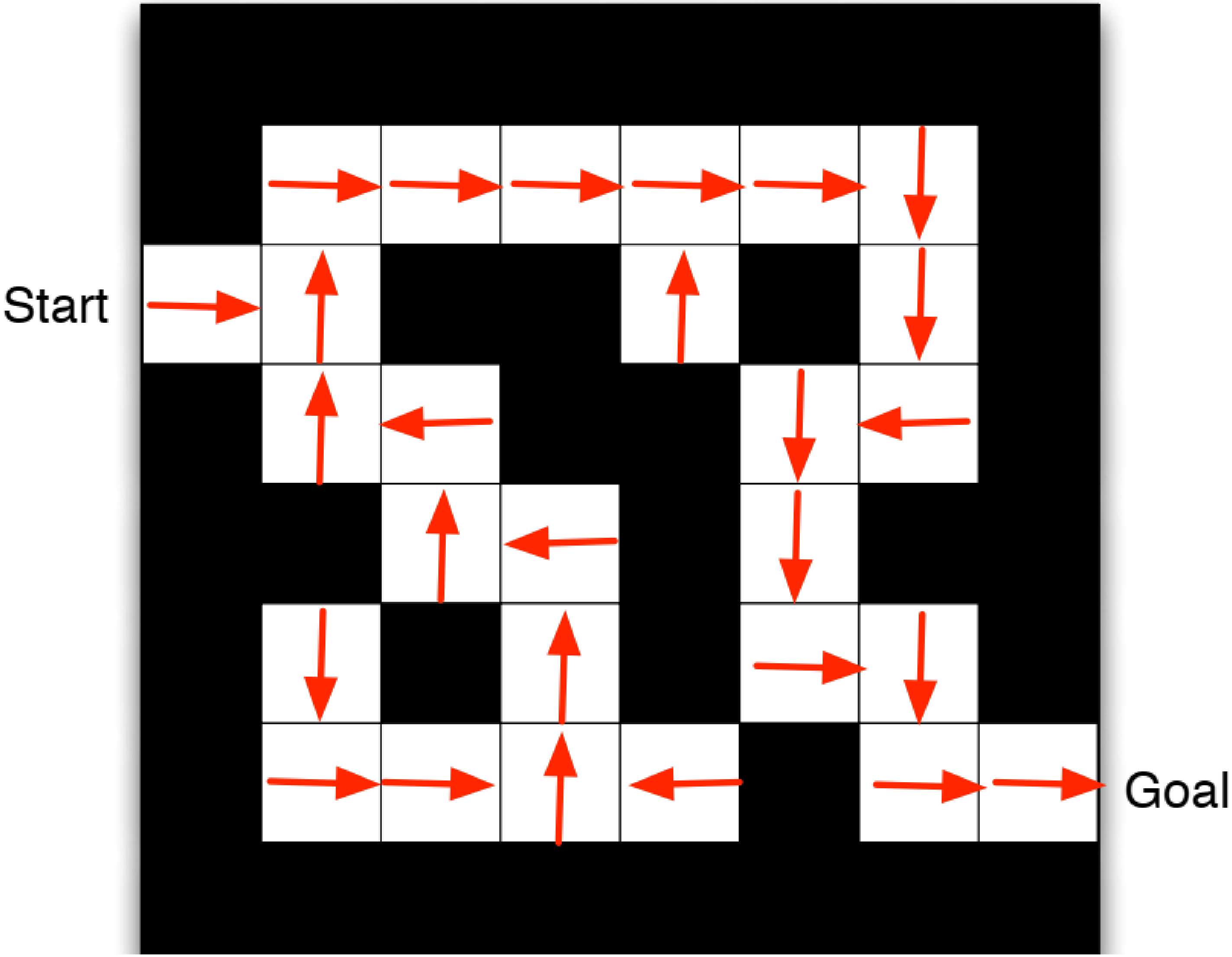
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The value of receiving reward R after $k + 1$ timesteps is $\gamma^k R$
- γ values immediate reward Vs delayed reward
  - $\gamma \approx 0$ leads to "myopic" evaluation
  - $\gamma \approx 1$ leads to "far-sighted" evaluation

Policy – At the core of an RL agent

- A policy π is the agent's behaviour
- It is a map from state s to action a
  - Deterministic policy: a $= \pi(s)$
  - Stochastic policy: $\pi(a|s) = P(A_t = a|S_t = s)$
- A policy π is a distribution over actions a given states

UNIVERSITÀ DI PISA



Maze Example (Policy)

Arrows represent policy $\pi(s)$ for each state s

Model

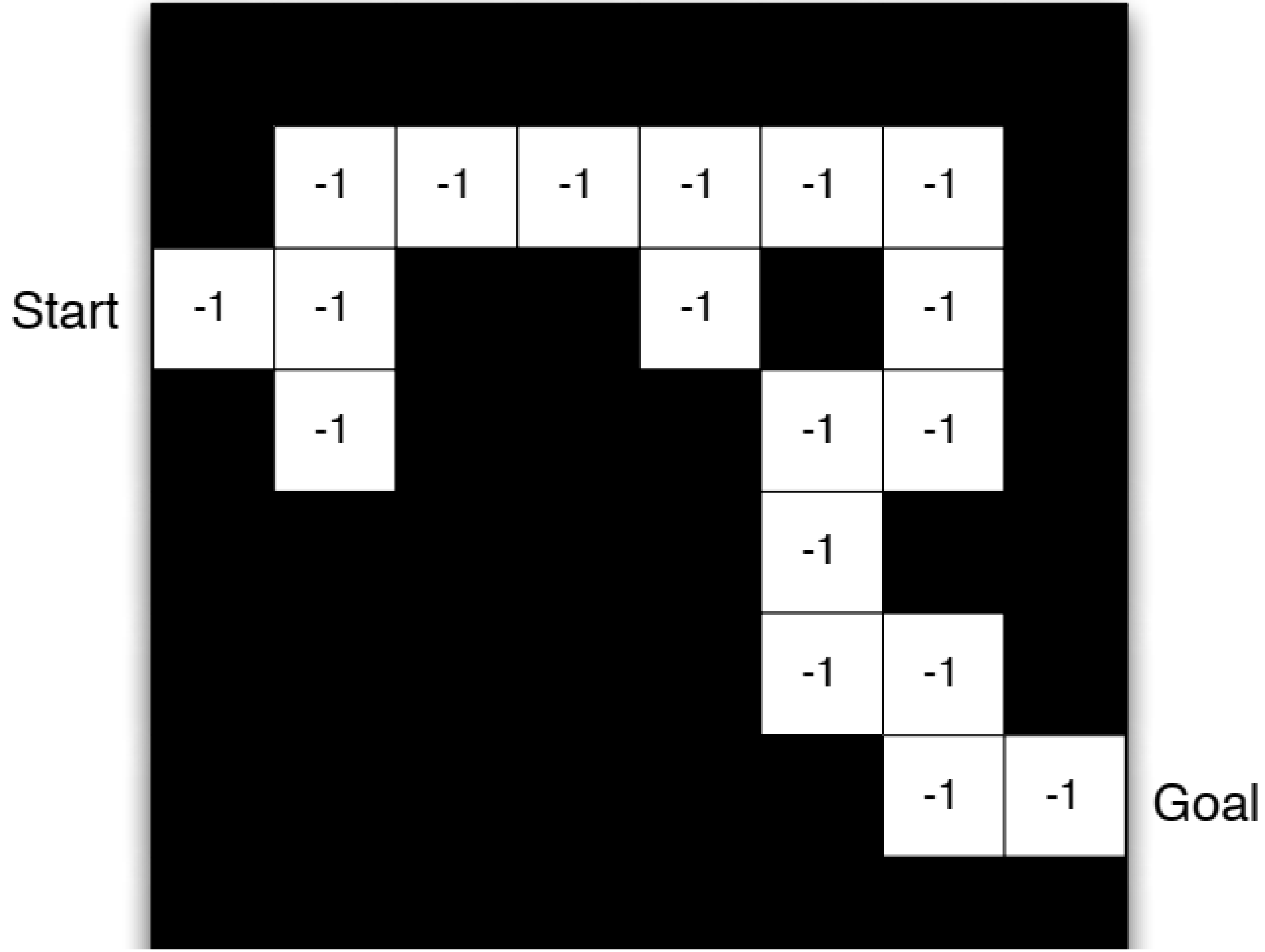
- A model predicts what the environment will do next
- Predict next state s' following an action a

$$\mathcal{P}_{ss'}^{a} = P(S_{t+1} = s' | S_t = s, A_t = a)$$

- Predict next reward

$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

Maze Example (Model)



- Agent may have an internal (imperfect) model of the environment
  - How actions change the state
  - How much reward from each state
- Grid Layout: transition model $\mathcal{P}_{ss'}^{a}$
- Numbers: immediate reward model $\mathcal{R}_{s}^{a}$

Value Function

- The state-value function $v_\pi(s)$ of a Markov Decision Process is the expected return starting from state and following policy π

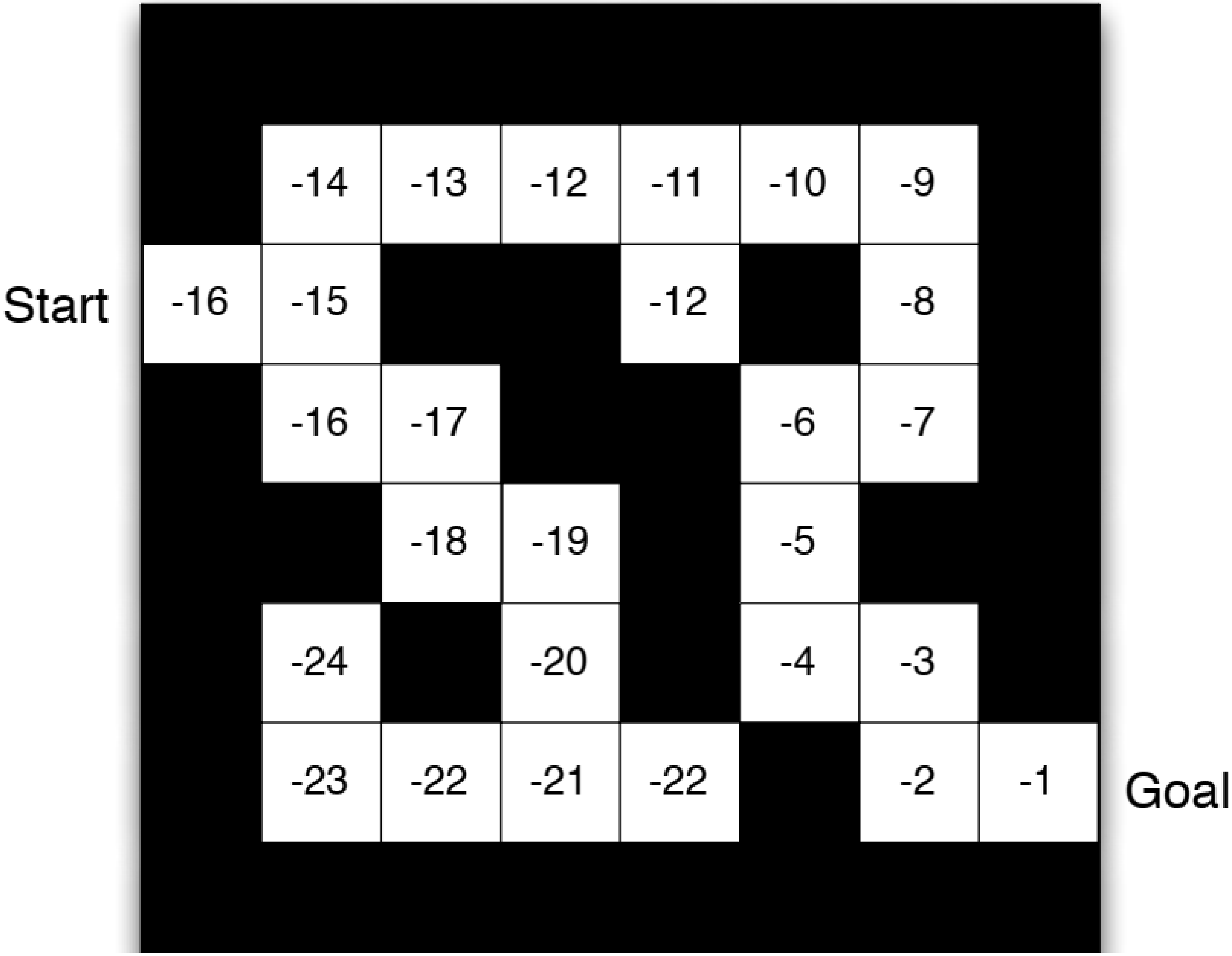
$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- The value function $v_\pi(S_t)$ can be decomposed into two parts
  - Immediate reward R_{t+1}
  - Discounted value of successor state $\gamma v_\pi(S_{t+1})$

$$v_\pi(s) = \mathcal{R}_s + \gamma \sum_{s'} P_{ss'} v_\pi(s')$$

The expected state-value of being in any state reachable from s

UNIVERSITÀ DI PISA



Maze Example (Value Function)

Numbers denote the value $v_{\pi}(s)$ for each s

Expected time to reach the goal

UNIVERSITÀ DI PISA

Action-Value Function

The action-value function $q_\pi(s,a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_\pi(s')$$

Also the value function can be written in terms of the action-value function

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Bellman Expectation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_\pi(s') \right)$$

The expected return of being in a state reachable from s through action a and then continue following policy

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

The expected return of any action a' taken from states reachable from s through action a (and then follow policy)

Finding an Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we straightforwardly find the optimal policy

Bellman Optimality Equations

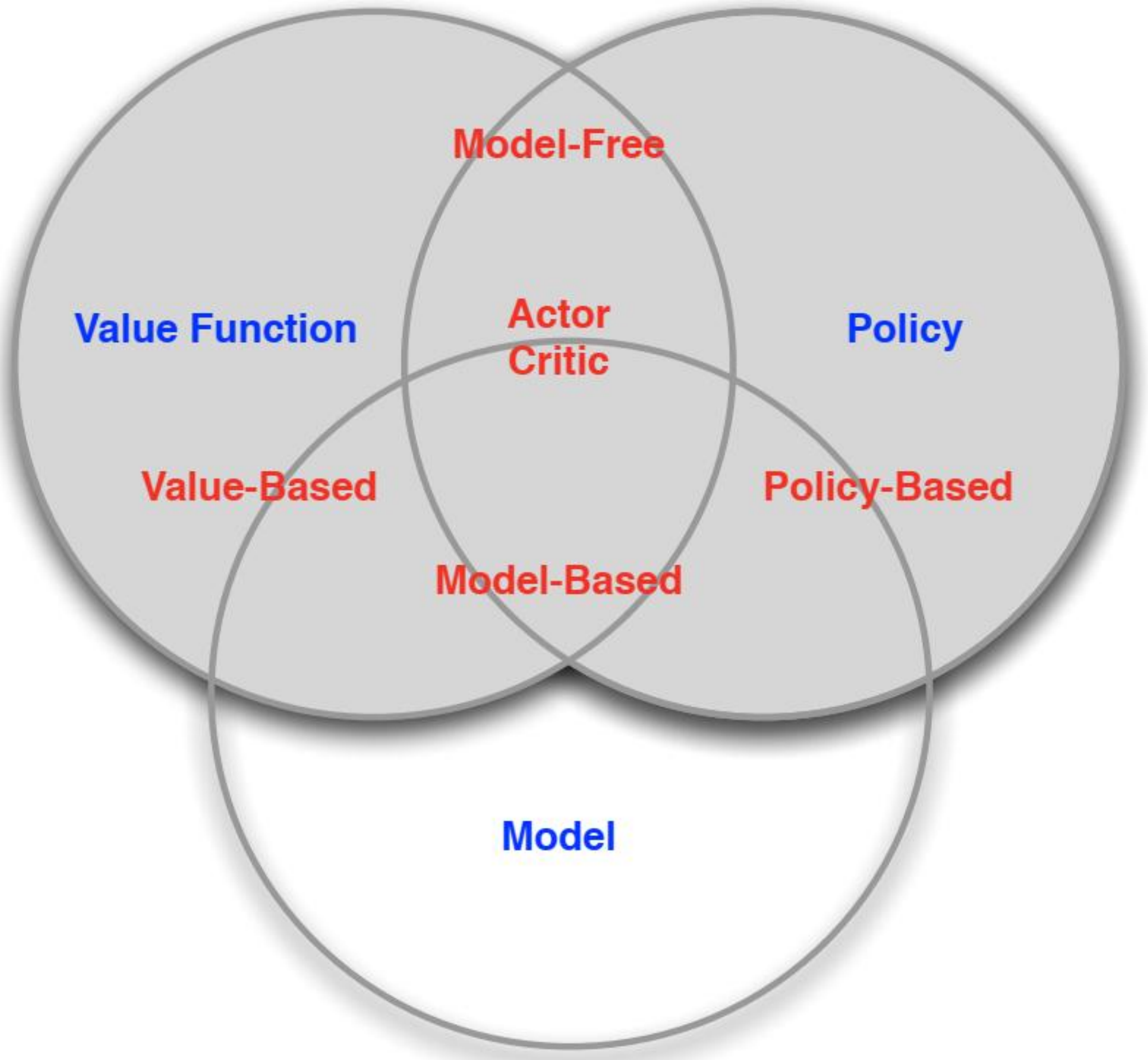
Optimal value functions are recursively related Bellman-style

$$v_*(s) = \max_{a \in \mathcal{A}} q_*(s,a) = \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_*(s')$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_*(s') = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a \max_{a' \in \mathcal{A}} q_*(s', a')$$

RL approaches

A Taxonomy

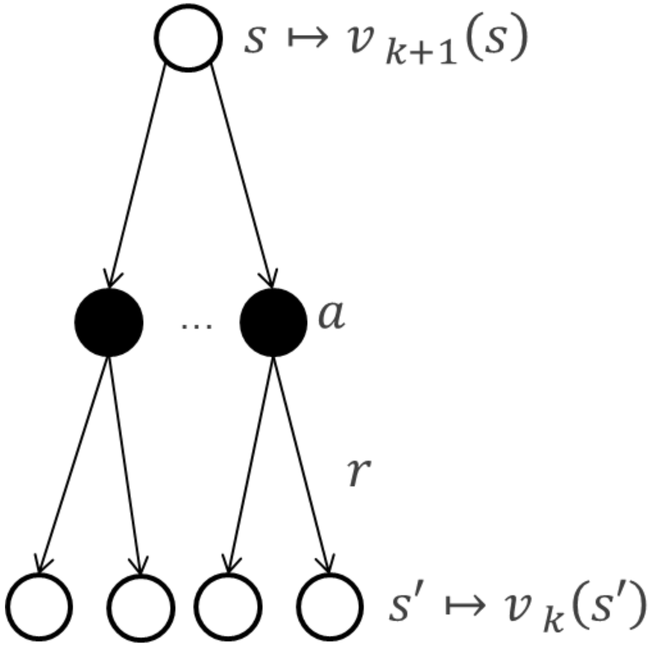

Model-Free
Value Function
Actor
Critic
Policy
Value-Based
Policy-Based
Model-Based
Model

Model Based - Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup

$$v_1 \to v_2 \to \cdots \to v_\pi$$

- Using synchronous backups
  1. At each iteration $k+1$
  2. For all states $s \in \mathcal{S}$
  3. Update $v_{k+1}(s)$ from $v_k(s')$ where s' is a successor state of s



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_k(s') \right)$$

Model Based – Policy Iteration

- Given policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = greedy(\pi) \Rightarrow \pi'(s) = \arg\max_{a \in A} q_\pi(s, a)$$

- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to the optimal policy π_*

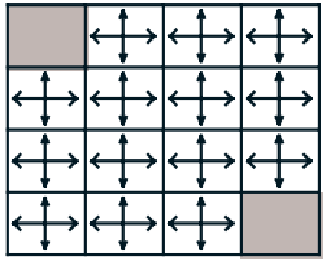
IN SUPREMÆ DIGNITATIS 1343 UNIVERSITÀ DI PISA

Policy Iteration Example (I)

v_k — Greedy policy on v_k

$k = 0$

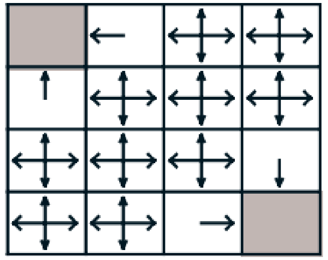
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



← random policy

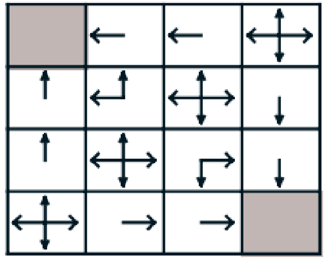
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Policy Iteration Example (II)

$k = 3$

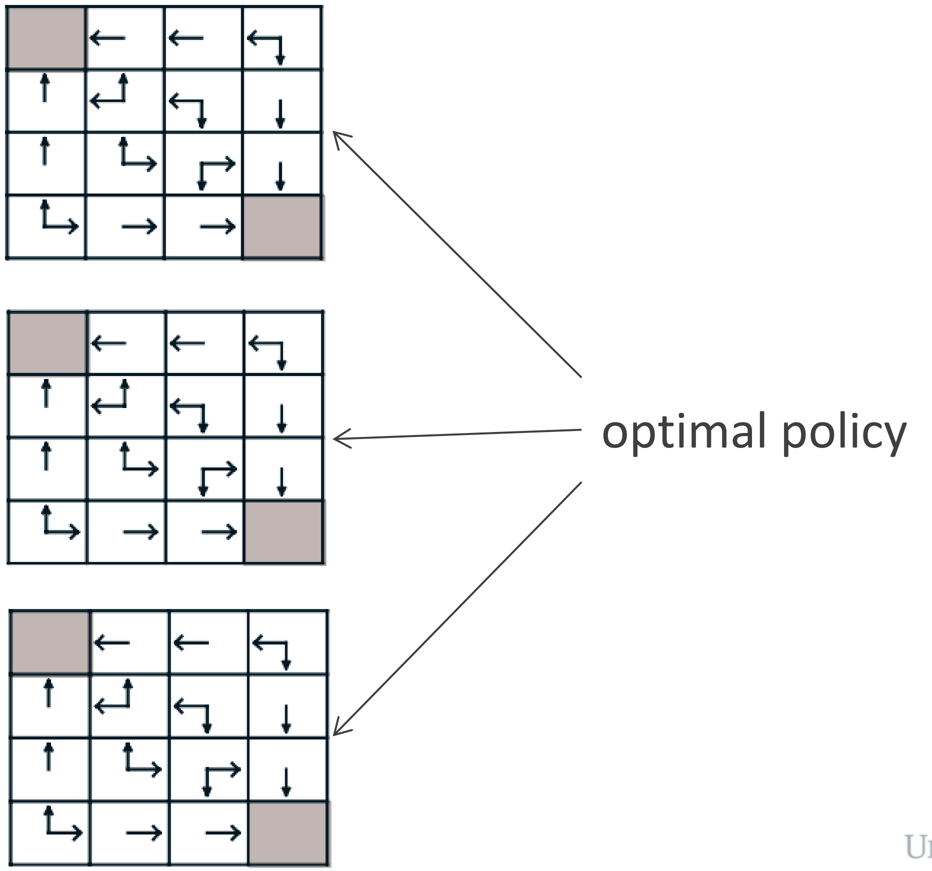
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

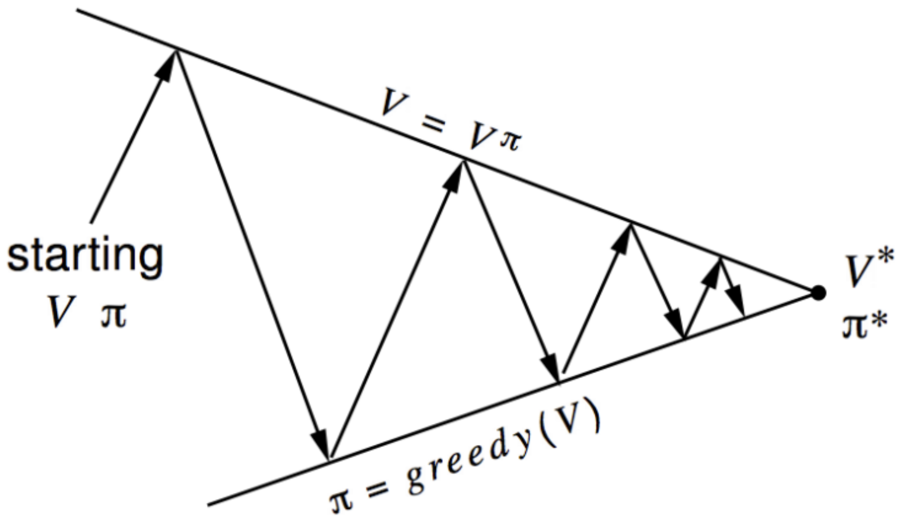
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

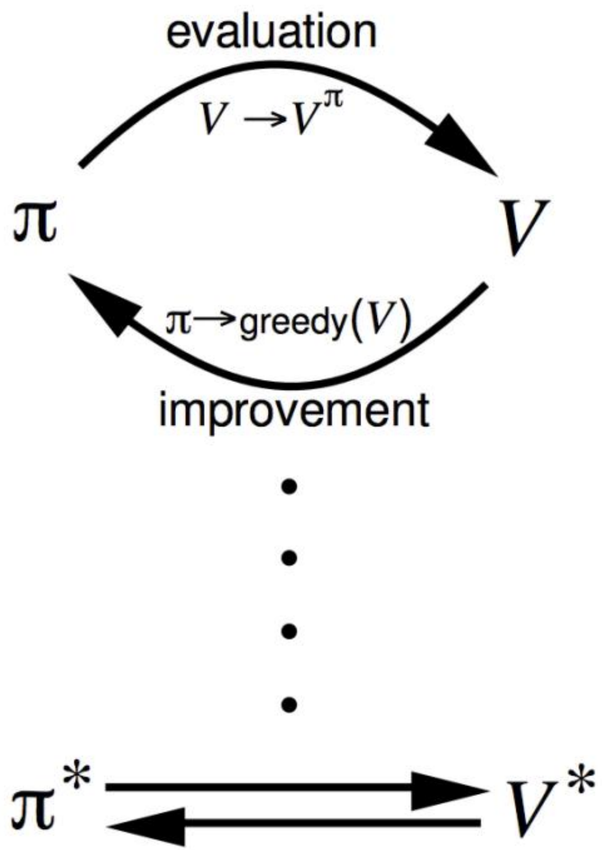
$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



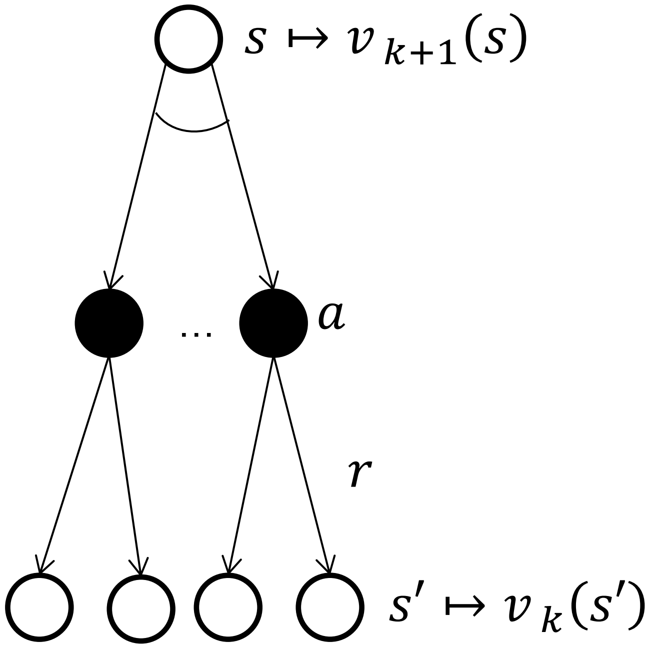
Policy Iteration





- ✓Policy evaluation - Estimate v_π
  - ✓Iterative policy evaluation
- ✓Policy improvement - Generate $\pi' \geq \pi$
  - ✓Greedy policy improvement

Model Based - Value Iteration



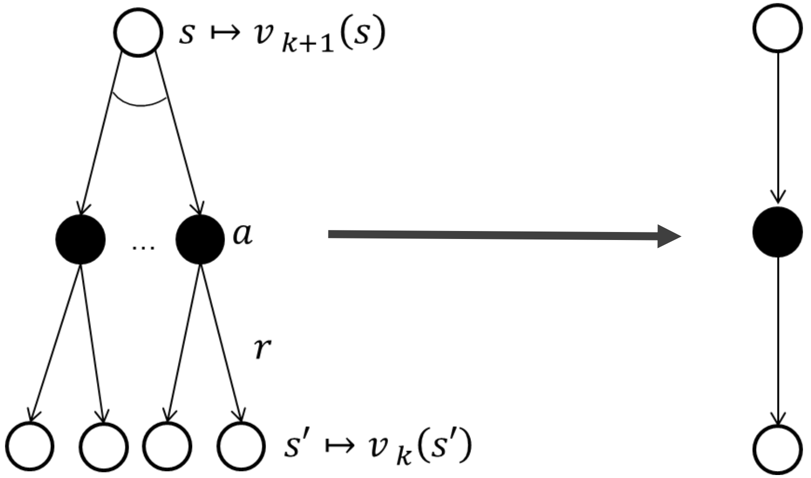
Using Bellman optimality in place of Bellman expectation

$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a v_k(s') \right)$$

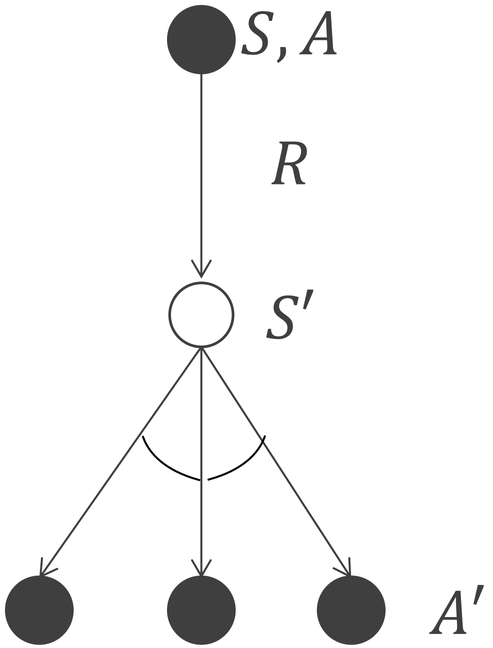
Model-Free Reinforcement Learning

- So far: solve a known MDP (states, transition, rewards, actions)
- Model free
  - No environment model
  - No knowledge of MDP transition/rewards
- Solution is to use sample updates

Using sample rewards and sample transitions $\langle S, A, R, S' \rangle$



Q-Learning – Off-policy RL



Greedy policy improvement on Q(S,A) is model-free

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \max_{a'} \gamma Q(S',a') - Q(S,A) \right)$$

Temporal difference error

- The target policy π is greedy w.r.t. $Q(S,A)$

$$\pi(S) = \arg\max_{a'} Q(S',a')$$

- Off policy - We choose which action A to execute based on an ϵ-greedy policy

$$\pi'(a|s) = \begin{cases} \epsilon/m + (1-\epsilon) & \text{if } a^* = \arg\max_{a \in \mathcal{A}} Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

Q-Learning Algorithm
for Off-policy Control

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until S is terminal

UNIVERSITÀ DI PISA

Value Function Approximation

So far $V(s)/Q(s,a)$ = lookup table

- An entry for every state s or state-action pair s, a
- Large MDPs $\Longrightarrow$ too many states and/or actions to store in memory
- Too slow to learn the value of each state individually
- Generalization issues

The *new* approach

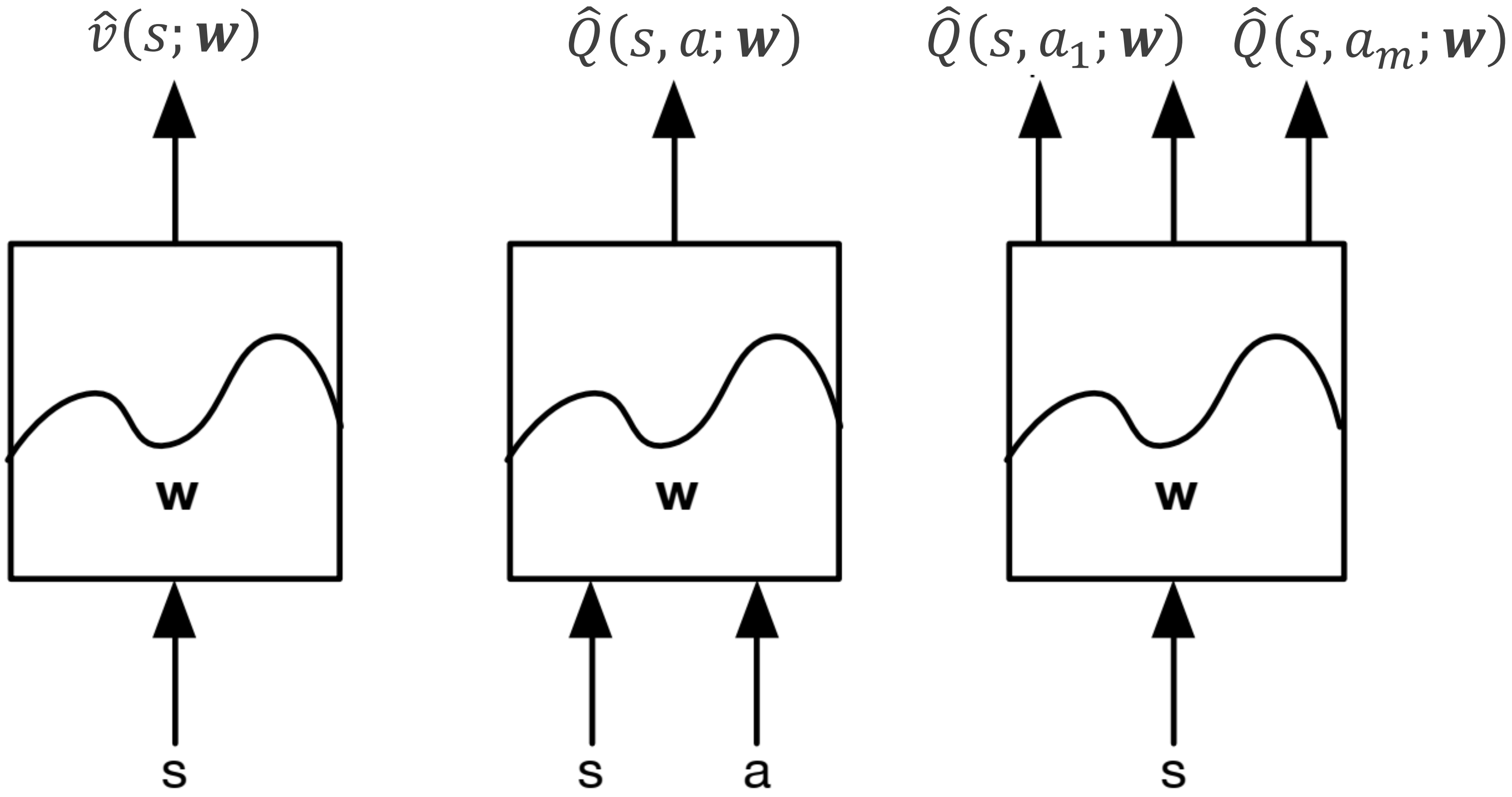
- Estimate value function with function approximation

$$\hat{v}(s; \boldsymbol{w}) \approx v_\pi(s)$$

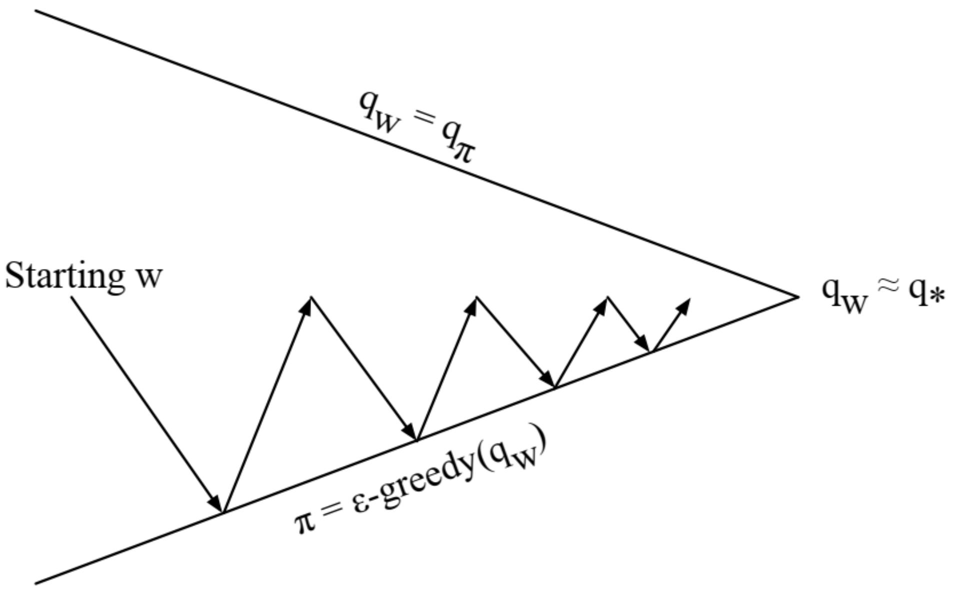
$$\hat{Q}(s, a; \boldsymbol{w}) \approx Q_\pi(s, a)$$

- Generalise from seen states to unseen states
- Update parameters $\boldsymbol{w}$ using Q-learning

Value Function Approximation Approaches



Learning with Value Function Approximation



- Policy evaluation - Approximate policy evaluation, $\hat{Q}(\cdot,\cdot;\boldsymbol{w}) \approx Q_\pi(\cdot,\cdot)$
- Policy improvement - ϵ-greedy policy improvement

Supervised Learning of Action-Value

Given a dataset of states and target temporal-difference targets

$$\mathcal{D} = \{\langle S_1, R_1 + \max_{a'} \gamma Q(S_1, a')\rangle, \langle S_2, R_2 + \max_{a'} \gamma Q(S_2, a')\rangle, \dots, \langle S_T, R_T + \max_{a'} \gamma Q(S_T, a')\rangle\}$$

Given a differentiable approximator $\hat{Q}(s, a; \boldsymbol{w})$ train it by SGD following

1. Sample state, value from experience

$$\langle s, Q^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta \boldsymbol{w} = \alpha\left(Q^\pi - \hat{Q}(s, a; \boldsymbol{w})\right)\nabla_{\boldsymbol{w}} \hat{Q}(s, a\,; \boldsymbol{w})$$

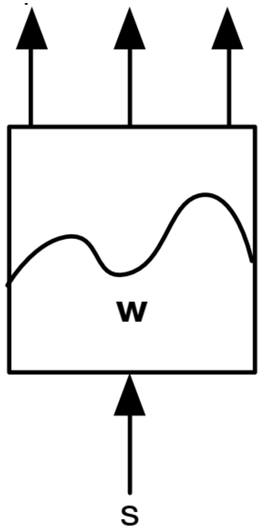
Deep Q-Networks (DQN)

- DQN uses experience replay and fixed Q-targets
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets with respect to old fixed parameters $\boldsymbol{w}^-$
- Optimise MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

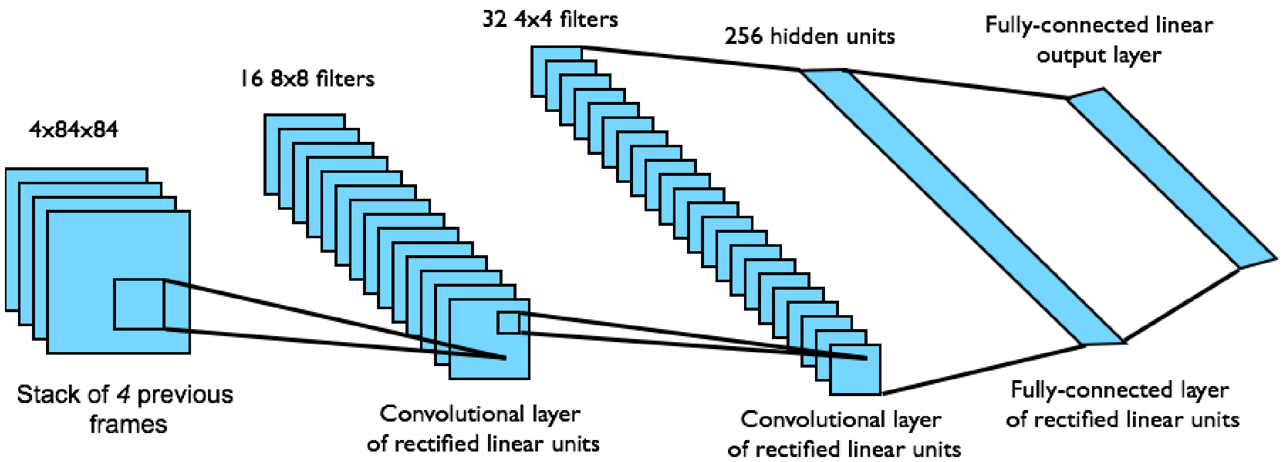
- Using variant of stochastic gradient descent

Atari-DQN



- End-to-end learning of values $Q(s,a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s,a)$ for 18 joystick/button positions
- Reward is change in score for that step

Network architecture and hyperparameters fixed across all games



UNIVERSITÀ DI PISA

Policy-Based Reinforcement Learning

Previously

- Approximate value or action-value function using parameters θ

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- Generate policy from the value function (e.g. using ϵ-greedy)

Now

- Parametrise the policy

$$\pi_\theta(s,a) = P(a|s,\theta)$$

- Focus again on model-free reinforcement learning

Policy-Based RL – Pros and Cons

- Advantages
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

Deep Policy Networks

- Represent policy by deep network with weights u

$$a = \pi_u(a|s) \text{ or } \pi_u(s)$$

- Define objective function as total discounted reward

$$J(u) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 \ldots |u]$$

- Optimise objective end-to-end by stochastic gradient descent
- Adjust policy parameters u to achieve more reward

Policy Gradient

How to make high-value actions more likely

- The gradient of a stochastic policy $\pi(a|s, \boldsymbol{u})$ is given by

$$\nabla_u J(u) = \mathbb{E}_\pi[\nabla_u \log \pi_u(a|s)\, Q^\pi(s, a)]$$

- The gradient of a deterministic policy a = $\pi(s)$ is given by

$$\nabla_u J(u) = \mathbb{E}_\pi[\nabla_a Q^\pi(s, a) \nabla_u a]$$

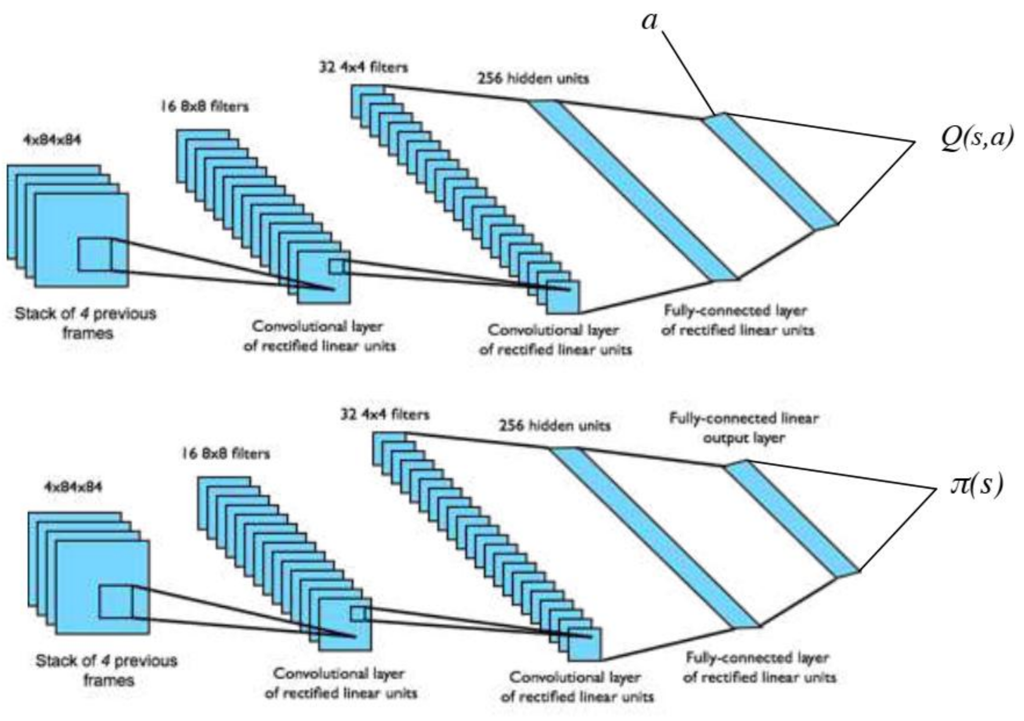
- Assuming a continuous and Q differentiable

(Deep) Actor-Critic Architectures

- Estimate value function $Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$
- Update policy parameters u by stochastic gradient ascent

$$\frac{\partial J(u)}{\partial u} = \frac{\partial \log \pi_u(a|s)}{\partial u} Q_w(s,a)$$

- Two separate CNNs are used for Q and π
- Policy π is adjusted in direction that most improves Q



Wrap Up

Useful Libraries

Gymnasium

A toolkit for developing and comparing reinforcement learning algorithms

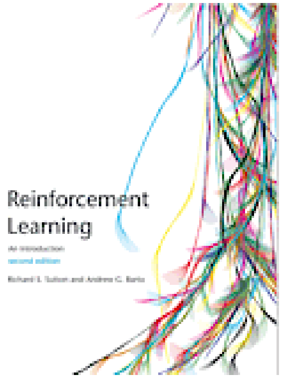
- ✓ Implementation of the interaction environment
- ✓ Plug-in your agent with integration of main DL frameworks

Stable Baselines

Reliable implementations of reinforcement learning algorithms in PyTorch

- ✓ Integration with Weights & Biases, Hugging Face and Gymnasium

A classic book if you want to know more



Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Second Edition, MIT Press (available online)

Take home messages

- Reinforcement learning is a general-purpose framework for decision-making
- MDP are a formalism to describe a fully-observable environment for RL
  - Can be relaxed to infinite and continuous actions/state and partially observable environments
- Model based - Solve a known MDP
  - Policy iteration - Re-define the policy at each step and compute the value according to this new policy until the policy converges
  - Value iteration - Computes the optimal state value function by iteratively improving the estimate of V(s)
- Model free - Optimise the value/policy of an unknown MDP
  - Value-based – Smoother learning task with deterministic policy
  - Policy-based – Faster convergence and stochastic policies
  - Actor-critic – Learn the value function to reduce variance of policy gradient
- …and much more (including planning with learned model (AlphaX))

Next Lecture

- Final lecture
  - With full exam information