

Deep Reinforcement Learning

Francisco S. Melo

Deep Structured Learning Course
25/11/2019

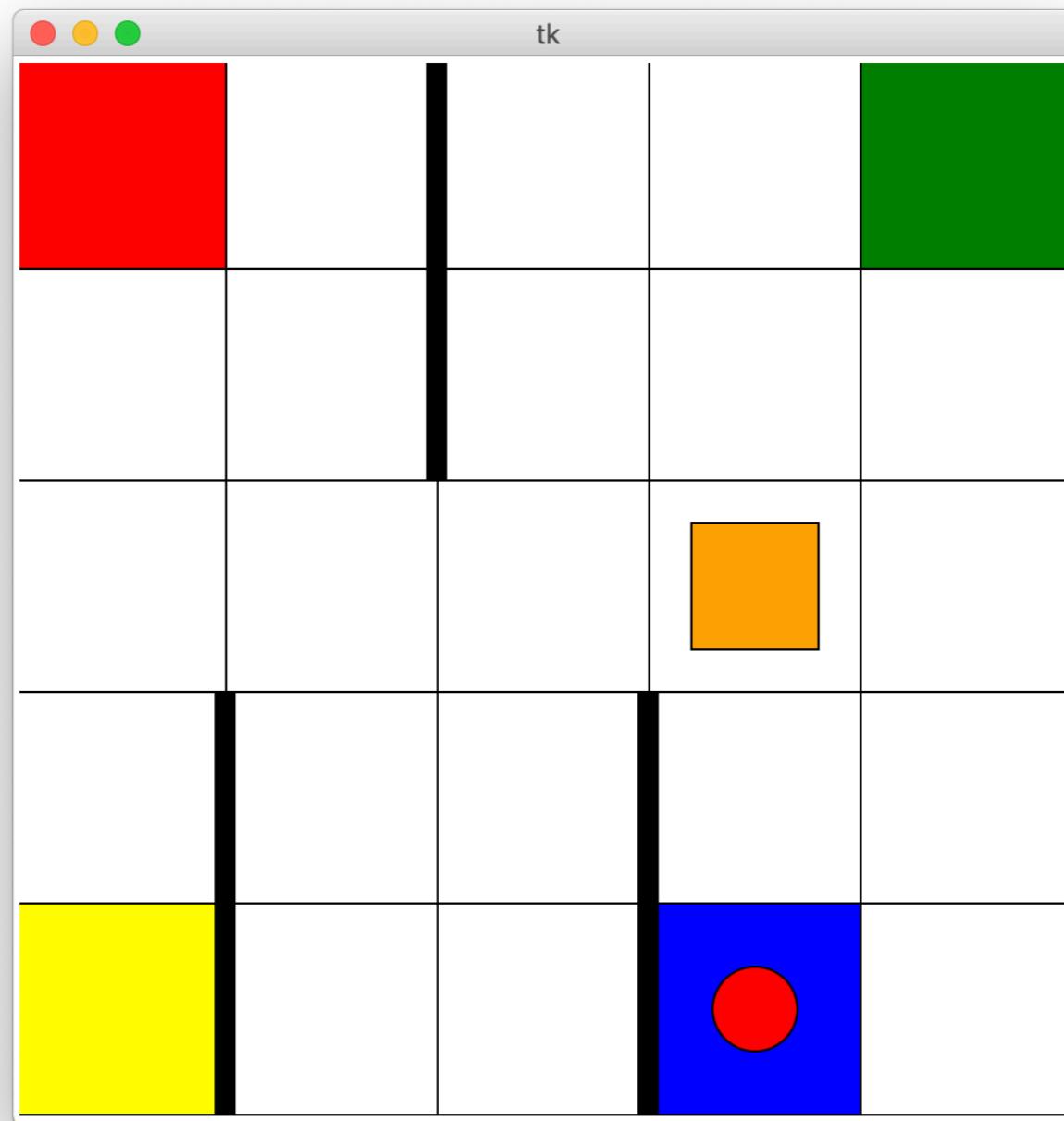
Outline of the lecture

- **Part I: RL Primer**
 - The RL Problem
 - Markov Decision Process - A Model for RL Problems
 - Optimality & Dynamic Programming
 - Monte Carlo Approaches
 - Temporal Difference Learning
 - The Policy Gradient Theorem

Outline of the lecture

- **Part II: Deep RL**
 - From RL to Deep RL
 - DQN
 - Deep advantage actor-critic methods
 - Trust region methods

The RL Problem



The RL Problem

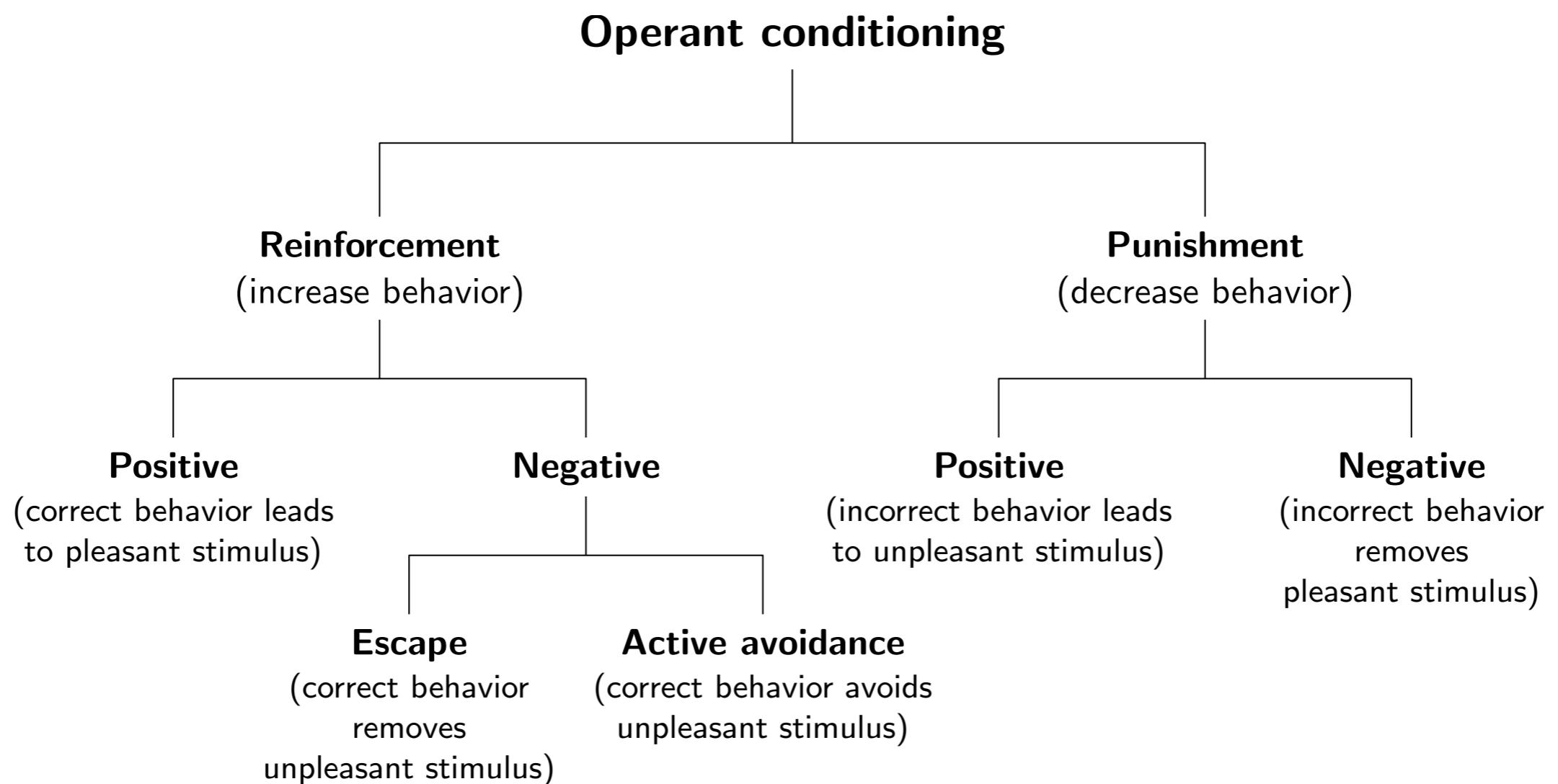
- Ingredients for success:
 - You learned as you played the game
 - You **experimented** the different actions
 - As soon as you figured out the goal of the game, you **stopped experimenting**
 - You used the **feedback** you got (n. of steps) to figure out the goal of the game
 - When pursuing the goal, you had to **think ahead** to select the actions

The RL Problem



What is RL?

- Inspired on theory of **operant conditioning**



What is RL?

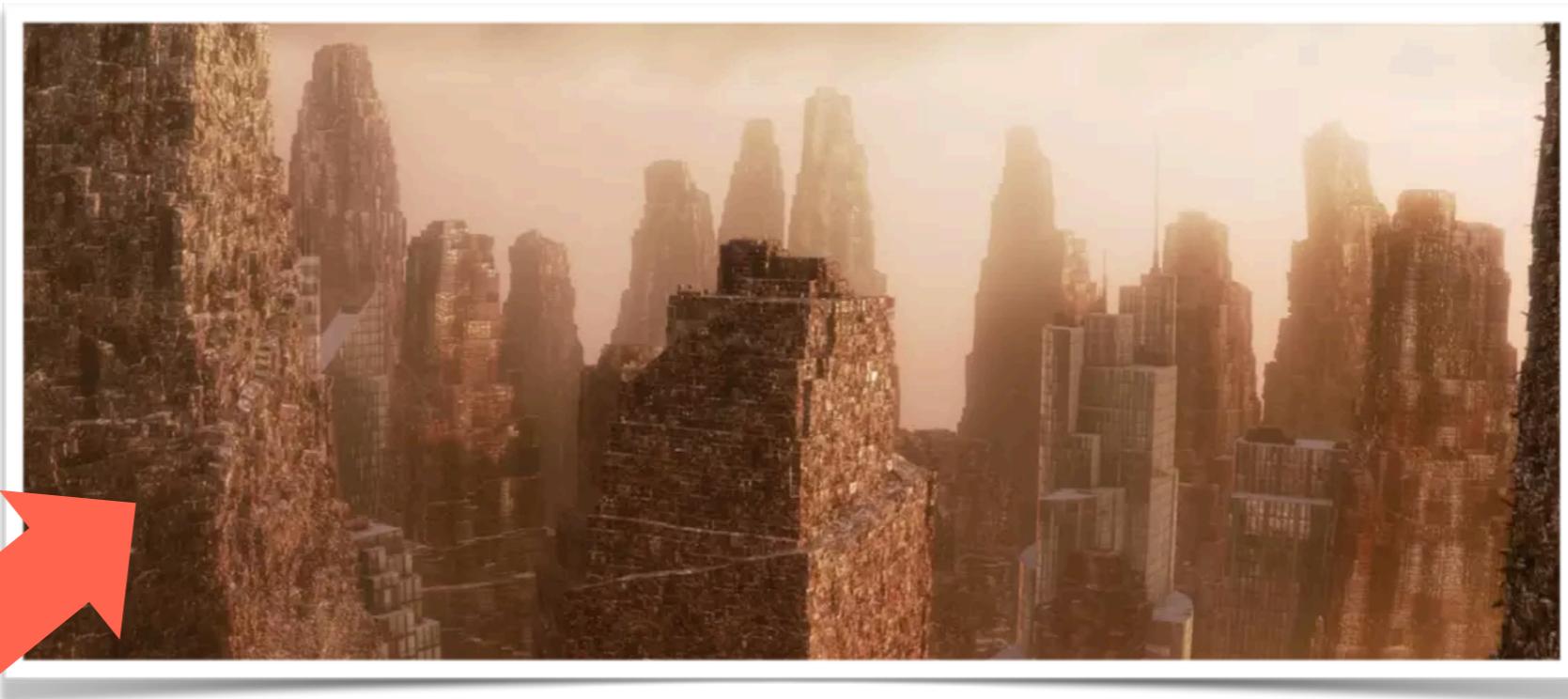
- Computational “counterpart” to operant conditioning
- Class of problems and algorithms to solve those problems
- Learning takes place through the interaction between agent and environment
(learning by trial-and-error)
- Learning driven by a “*reinforcement signal*” rather than examples

Elements in RL

- Key elements in RL:
 - Interactive learning
 - Learning from evaluative feedback
 - Tradeoff between exploration and exploitation
 - Actions impact the future (temporal credit assignment)

Interactive learning

Environment



Interaction



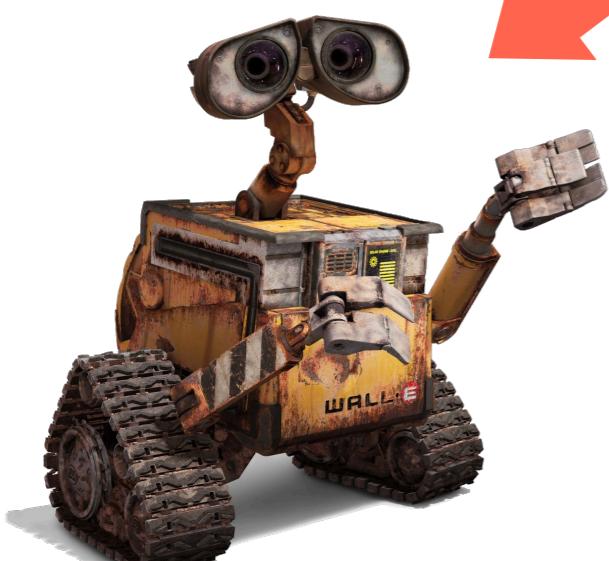
Agent

Interactive learning

Environment

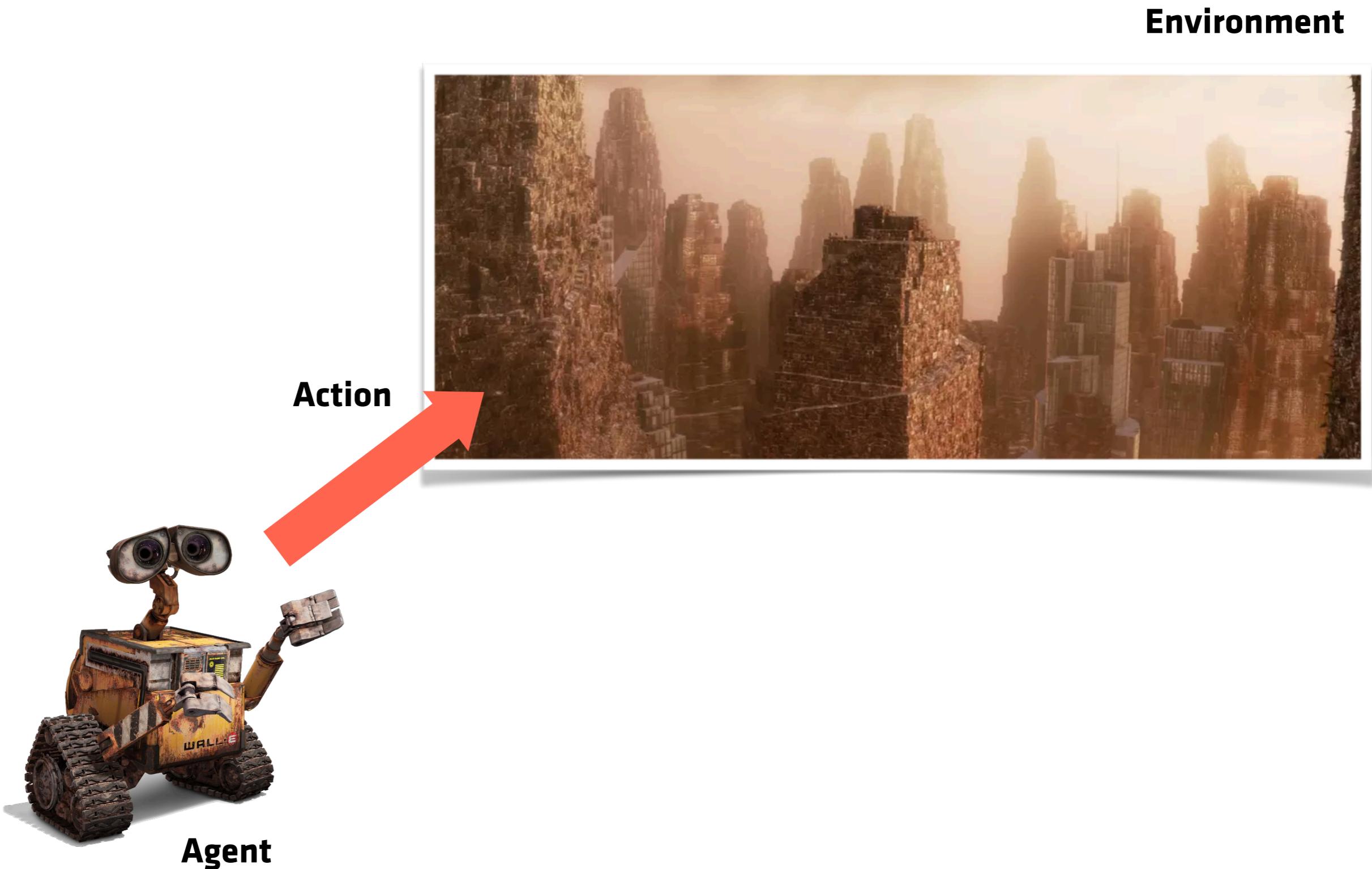


State



Agent

Interactive learning



Interactive learning

Environment may change state



Reward



Agent

Markov decision process

- Formalizing the reinforcement learning problem:
 - The **state** of the world/environment at step t is denoted as S_t
 - The state takes values in some set \mathcal{S} (the **state space**)

Markov decision process

- Formalizing the reinforcement learning problem:
 - The **action** of the agent at step t is denoted as A_t
 - The action takes values in some set \mathcal{A} (the **action space**)

Markov decision process

- Formalizing the reinforcement learning problem:
 - Upon performing an action at time step t , the agent gets a (random) reward R_t
 - The reward depends on the state S_t and action A_t as

$$\mathbb{E} [R_t] = r(S_t, A_t)$$

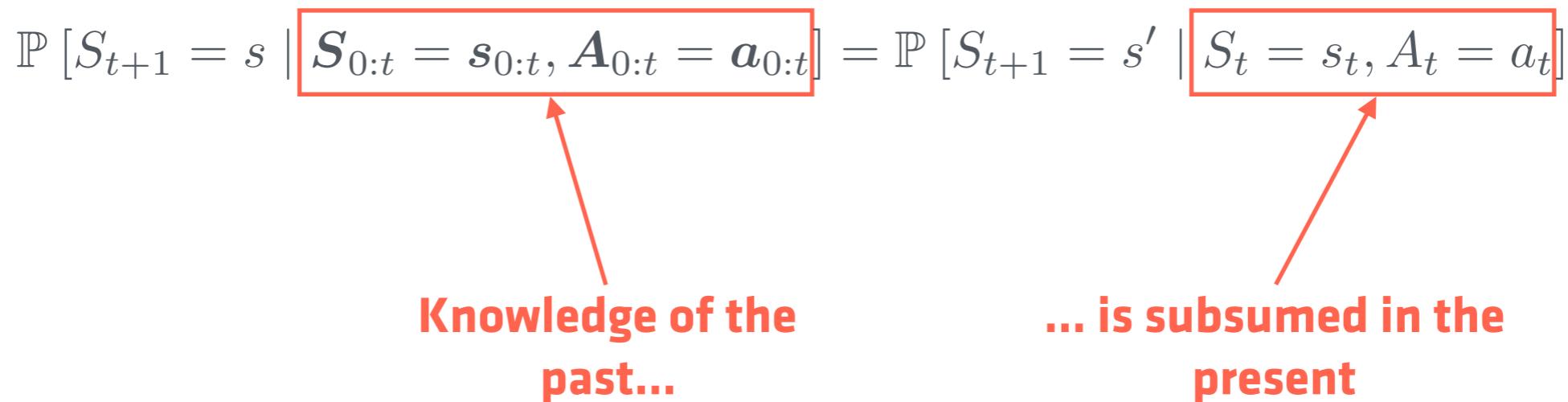
- We call r the **reward function**

Markov decision process

- Formalizing the reinforcement learning problem:
 - As a result of the agent's action at time step t , the state of the environment at time step $t + 1$ may change
 - We assume that the evolution of the state verifies the **Markov property**:

$$\mathbb{P} [S_{t+1} = s \mid \boxed{S_{0:t} = s_{0:t}, A_{0:t} = a_{0:t}}] = \mathbb{P} [S_{t+1} = s' \mid \boxed{S_t = s_t, A_t = a_t}]$$

Knowledge of the past... ... is subsumed in the present



Markov decision process

- Formalizing the reinforcement learning problem:
 - As a result of the agent's action at time step t , the state of the environment at time step $t + 1$ may change
 - We assume that the evolution of the state verifies the **Markov property**:

$$\mathbb{P} [S_{t+1} = s \mid S_{0:t} = s_{0:t}, A_{0:t} = a_{0:t}] = \mathbb{P} [S_{t+1} = s' \mid S_t = s_t, A_t = a_t]$$

- We call these the **transition probabilities**, and write

$$\mathbf{P}(s' \mid s, a) = \mathbb{P} [S_{t+1} = s' \mid S_t = s, A_t = a]$$

Markov decision process

- A **Markov decision process** is defined as a tuple $(\mathcal{S}, \mathcal{A}, \{\mathbf{P}_a, a \in \mathcal{A}\}, r)$
 - \mathcal{S} is the state space
 - \mathcal{A} is the action space
 - For each action $a \in \mathcal{A}$, \mathbf{P}_a is a matrix with entry ss' given by $\mathbf{P}(s' | s, a)$
 - r is the reward function

... so what?

Optimality

- A Markov decision **process** is not actually a **problem**
 - Provides a mere descriptive model for RL problems
 - What does it mean to solve a model??



Objective

Optimality

- We thus formulate a **Markov decision problem** (MDP) as follows:

Given a Markov decision process and a function

$$J(\{R_t, t = 0, \dots, \})$$

how can we select the actions $\{A_t\}$ to maximize J ?

Policies

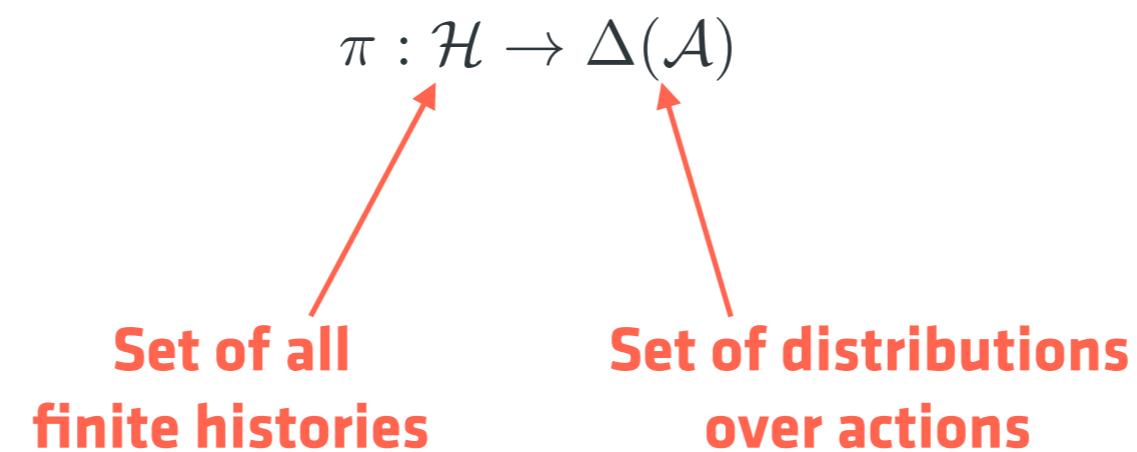
- MDPs are formulated in terms of **action selection**
- A **policy** is an “action selection rule”:
- Define the **history at time step t** as

$$H_t = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t\}$$

- It is a random variable
- Depends on the particular action selection

Policies

- A policy is a mapping π between histories and distributions over actions:

$$\pi : \mathcal{H} \rightarrow \Delta(\mathcal{A})$$


The diagram illustrates the function π . At the top, the mathematical expression $\pi : \mathcal{H} \rightarrow \Delta(\mathcal{A})$ is shown. Below it, two red arrows point upwards from descriptive text to the respective sets. The left arrow points from the text "Set of all finite histories" to the domain \mathcal{H} . The right arrow points from the text "Set of distributions over actions" to the codomain $\Delta(\mathcal{A})$.

Set of all finite histories

Set of distributions over actions

Policies

- **Types of policies:**

- **Deterministic policies** - Each history is mapped to exactly one action

$$\pi : \mathcal{H} \rightarrow \mathcal{A}$$

- **Markov policies** - Depend only on the most recent state (may be time-dependent)

$$\pi_t : \mathcal{S} \rightarrow \Delta(\mathcal{A})$$

- **Stationary policies** - Depend only on the most recent state (is time-independent)

$$\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$$

Optimality criteria

- J in the previous formulation is the **optimality criterion**
- There are several possible optimality criteria in the literature
 - Each has advantages and disadvantages
 - The choice should be problem-driven

Optimality criteria

- **(Expected) immediate reward:**

$$J(\{R_t, t = 0, \dots, \}) = \mathbb{E} [R_t] = r(S_t, A_t)$$

- Advantages:
 - Simple to optimize:
$$\pi(S_t) = \operatorname{argmax}_{a \in \mathcal{A}} r(S_t, a)$$
- Disadvantages:
 - Only applicable in very specific problems

Optimality criteria

- **(Expected) total reward:**

$$J(\{R_t, t = 0, \dots, \}) = \mathbb{E} \left[\sum_{t=0}^{\infty} R_t \right]$$

- Advantages:
 - Not myopic
- Disadvantages:
 - Objective not always well-defined (summation may diverge)

Optimality criteria

- **(Expected) average per-step reward:**

$$J(\{R_t, t = 0, \dots, \}) = \lim_{T \rightarrow \infty} \frac{1}{T} \mathbb{E} \left[\sum_{t=0}^T R_t \right]$$

- Advantages:
 - Not myopic
 - Independent of initial state of the process
- Disadvantages:
 - Sometimes cumbersome to work with

Optimality criteria

- (Expected) total discounted reward:

$$J(\{R_t, t = 0, \dots, \}) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

- Advantages:

- Not myopic
- “Economical” interpretation

- Disadvantages:

- Depends on the initial state of the process

Discount
 $0 \leq \gamma < 1$

We henceforth focus
on this criterion

Markov decision problem (MDP)

- A **Markov decision problem** is defined as a tuple $(\mathcal{S}, \mathcal{A}, \{\mathbf{P}_a, a \in \mathcal{A}\}, r, \gamma)$
 - \mathcal{S} is the state space
 - \mathcal{A} is the action space
 - For each action $a \in \mathcal{A}$, \mathbf{P}_a is a matrix with entry ss' given by $\mathbf{P}(s' \mid s, a)$
 - r is the reward function
 - γ is the discount

Solving MDPs

Value function

- Let us consider a fixed **stationary** policy π
 - Action depends only on current state
 - Invariant through time
- In other words,

$$\pi(a \mid s) = \mathbb{P}[A_t = a \mid S_t = s]$$



Independent of t

Value function

- The value of J depends on the initial state
- Let

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, \right]$$

- $v_\pi(s)$ is the value of J when
 - The agent follows policy π , i.e.,

$$A_t \sim \pi(\cdot \mid S_t)$$

- The initial state is s

Value function

- The function

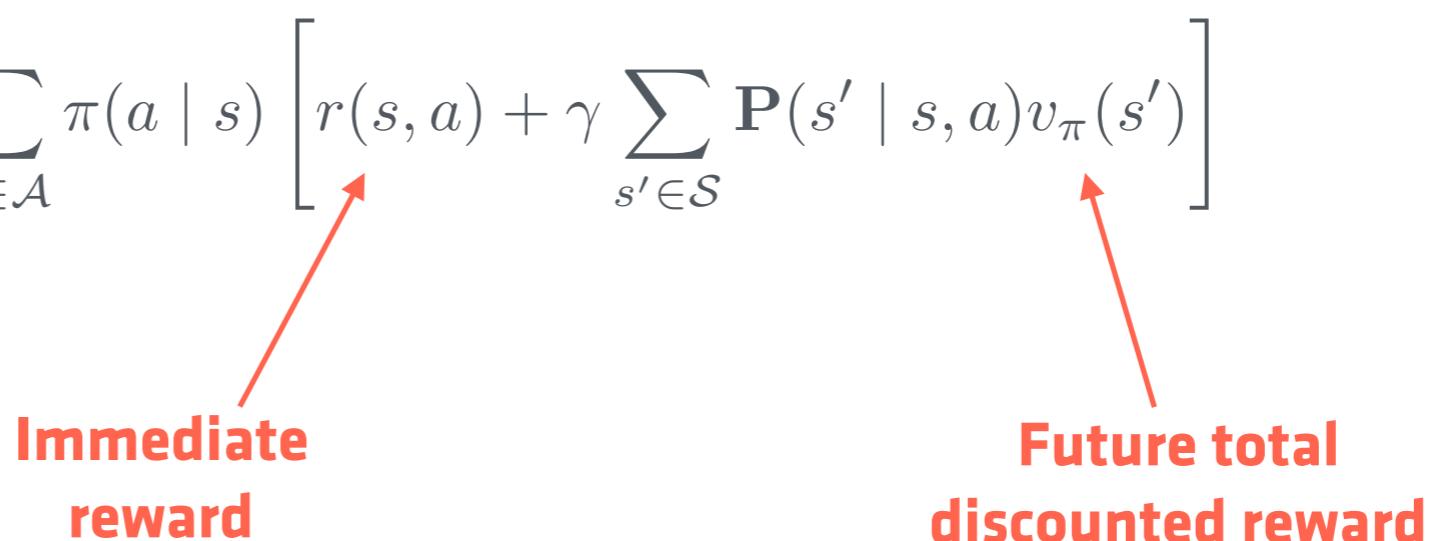
$$v_\pi : \mathcal{S} \rightarrow \mathbb{R}$$

is called a **value function**

- It is the **value function associated with π**
- It verifies the recursive relation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) v_\pi(s') \right]$$

Immediate reward **Future total discounted reward**



A computational (parenthesis)

- The relation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v_\pi(s') \right]$$

offers two possibilities to compute v_π

- Solve the associated (linear) system of equations
- Starting with an arbitrary initial estimate $v^{(0)}$, repeatedly go over the update

$$v^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v^{(k)}(s') \right]$$

A computational (parenthesis)

- The iterative approach with update

$$v^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v^{(k)}(s') \right]$$

is known as **value iteration**

- Computing the value function associated with a policy is usually referred as the **prediction problem**
- It is a **dynamic programming** approach that, intuitively, “propagates” reward information back through time



... moving on...

Optimal policy

- We say that a policy π^* is **optimal** if and only if

$$v_{\pi^*}(s) \geq v_{\pi}(s), \forall \pi, \forall s \in \mathcal{S}$$

- That such a policy exists is a central result in the theory of MDPs



Solving MDP = Computing an optimal policy

Value function 2.0

- The value function for the (an) optimal policy is simply denoted as v^*
- It verifies the recursive relation

$$v^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^*(s') \right]$$

- The optimal policy can be computed from v^* as

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^*(s') \right]$$

A computational (parenthesis) 2.0

- The relation

$$v^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^*(s') \right]$$

also offers a possibility to compute v^*

- Starting with an arbitrary initial estimate $v^{(0)}$, repeatedly go over the update

$$v^{(k+1)}(s) \leftarrow \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^{(k)}(s') \right]$$

- An MDP can thus be solved by computing v^* (and π^* from it)

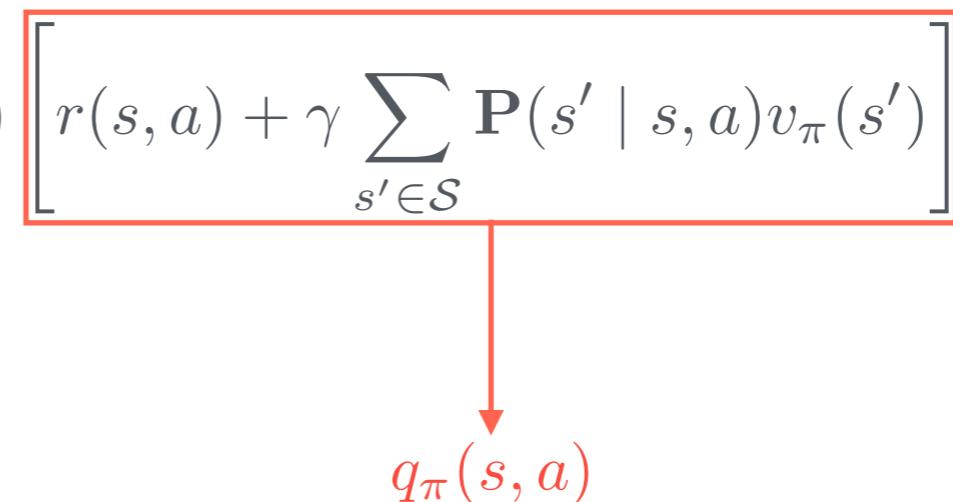


...

Value function 3.0

- **Other useful value functions to be considered**

- Action-value function (or Q -function) associated with a policy:

$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v_{\pi}(s') \right]$$

$$q_{\pi}(s, a)$$

Value function 3.0

- **Other useful value functions to be considered**

- Action-value function (or Q -function) associated with a policy:

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v_{\pi}(s')$$

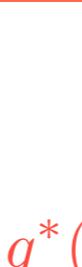
- It verifies the recursive relation

$$q_{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) \sum_{a' \in \mathcal{A}} \pi(a' | s') q_{\pi}(s', a')$$

Value function 3.0

- **Other useful value functions to be considered**

- Optimal action-value function (or Q -function):

$$v^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^*(s') \right]$$

$$q^*(s, a)$$

Value function 3.0

- **Other useful value functions to be considered**

- Optimal action-value function (or Q -function):

$$q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v^*(s')$$

- It verifies the recursive relation

$$q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) \max_{a' \in \mathcal{A}} q^*(s', a')$$

- Moreover,

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} q^*(s, a)$$

■ ■ ■

- We can compute q_π and q^* using similar iterative approaches

$$q^{(k+1)}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) \max_{a' \in \mathcal{A}} q^{(k)}(s', a')$$

$$q^{(k+1)}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) \sum_{a' \in \mathcal{A}} \pi(a' \mid s') q^{(k)}(s', a')$$

which are all collectively known as **value iteration**

- Computing the optimal Q-function is usually referred as the **control problem**

Value function 3.0

- **Other useful value functions to be considered**
 - Advantage function associated with a policy:
$$\text{adv}_\pi(s, a) = q_\pi(s, a) - v_\pi(s)$$
 - The advantage function does not verify a recursive relation

Wrap up

Key players in RL

- **Immediate reward**
 - Translates the goal of the agent
 - Instantaneous / myopic
- **Policy**
 - Action selection rule
 - Solving an MDP consists in finding the optimal policy

Key players in RL

- **Value function**
 - “Secondary” reward
 - Long-term evaluation of the states
 - Can be used to compute the policy
- **Model (Markov decision process)**
 - Description of the dynamics of the process (transition probabilities)

Solving RL

- Solving an RL problem consists of solving the associated MDP
 - Solving an MDP consists of computing the optimal policy.
 - E.g.,
 - Use value iteration to compute v^*
- or
- Use value iteration to compute q^*
 - Use any of the above to compute π^*

Outline of the lecture

- **Part I: RL Primer**

- The RL Problem
- Markov Decision Process - A Model for RL Problems
- Optimality & Dynamic Programming
- Monte Carlo Approaches
- Temporal Difference Learning
- The Policy Gradient Theorem

Reinforcement learning

Reinforcement learning

- Interaction between the agent and the environment
 - Agent observes that $S_t = s$
 - Agent performs an action $A_t = a$
 - Agent gets a reward R_t
 - At the next time step, agent observes $S_{t+1} = s'$
 - ...

Reinforcement learning

- At each step, the agent collects a **sample**, consisting of a tuple

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

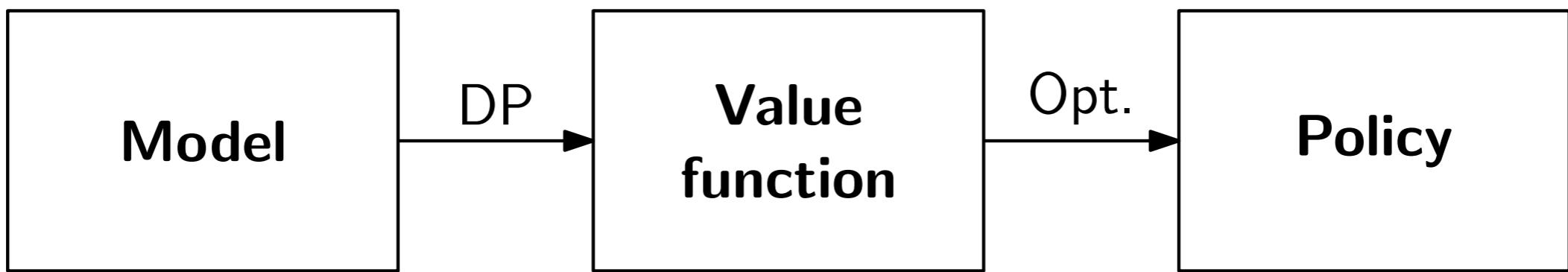
- Each such sample includes information about:
 - The reward, in the triplet (s, a, r)
 - The dynamics, in the triplet (s, a, s')

Reinforcement learning

- We consider explicitly the two subproblems within RL:
 - The **prediction problem** (given a policy, compute v_π)
 - The **control problem** (compute π^* – often by computing q^*)

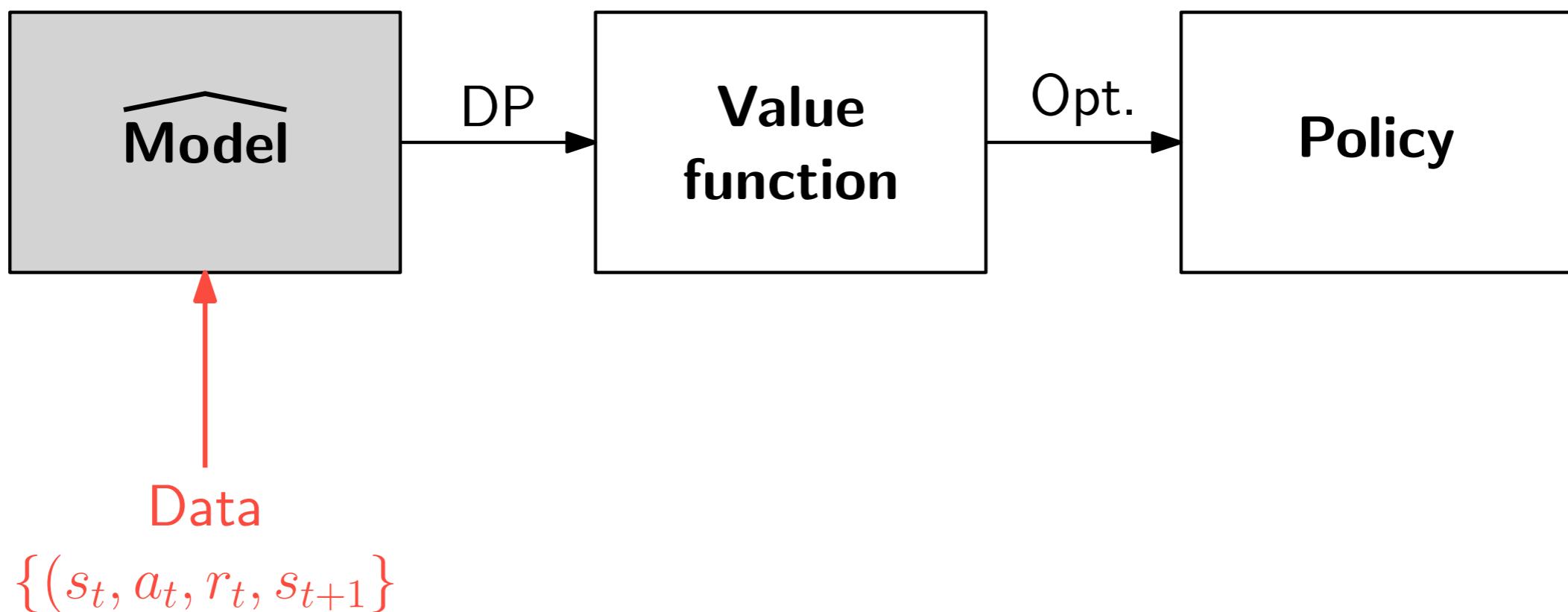
Taxonomy of RL methods

- Solving an MDP:



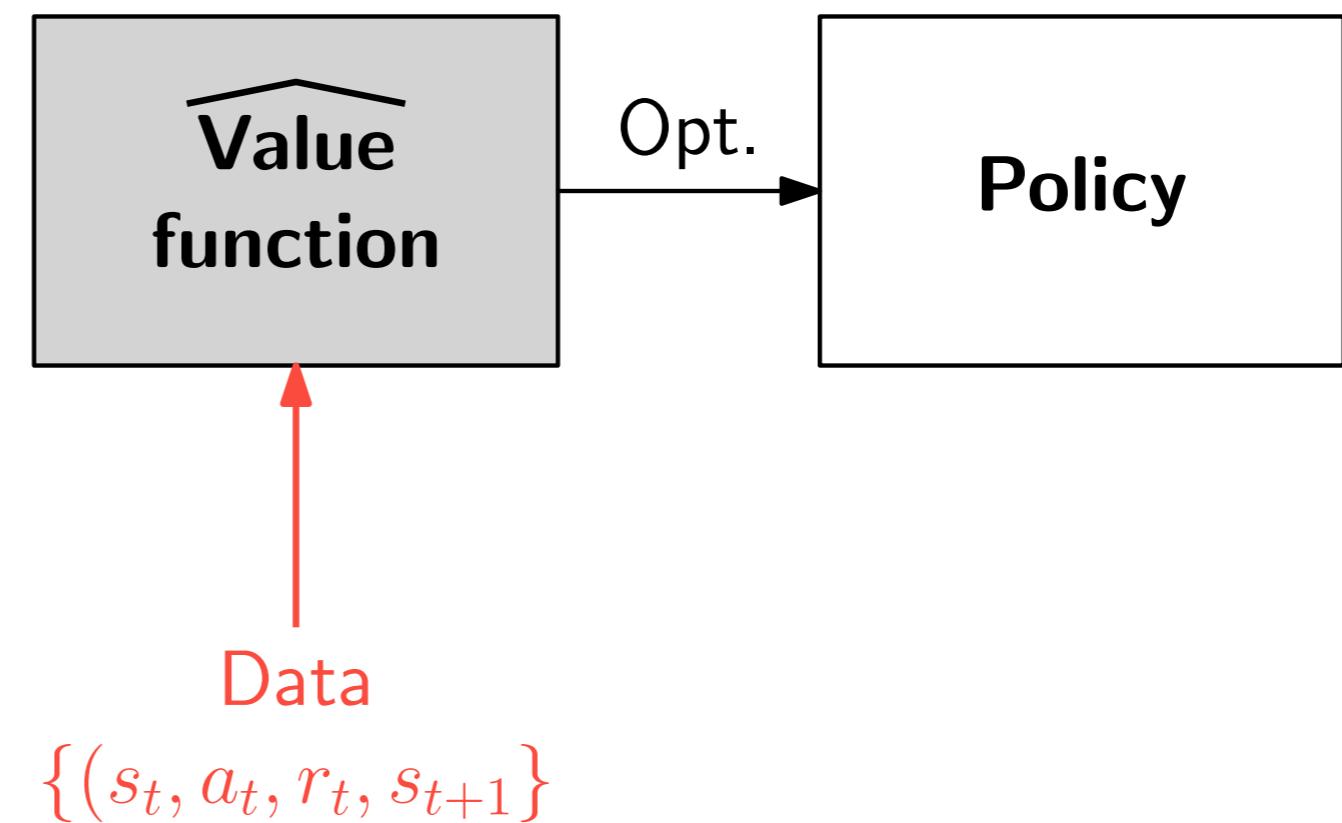
Taxonomy of RL methods

- Model-based methods:



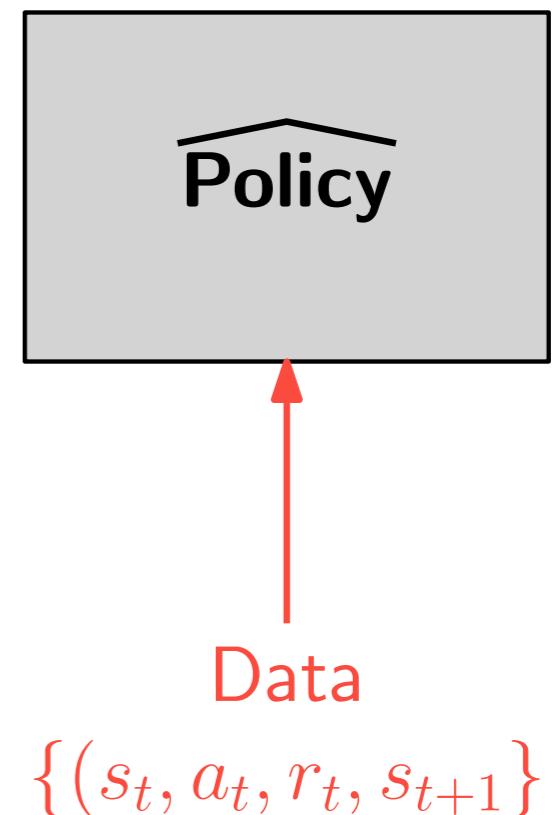
Taxonomy of RL methods

- **Value-based methods:**

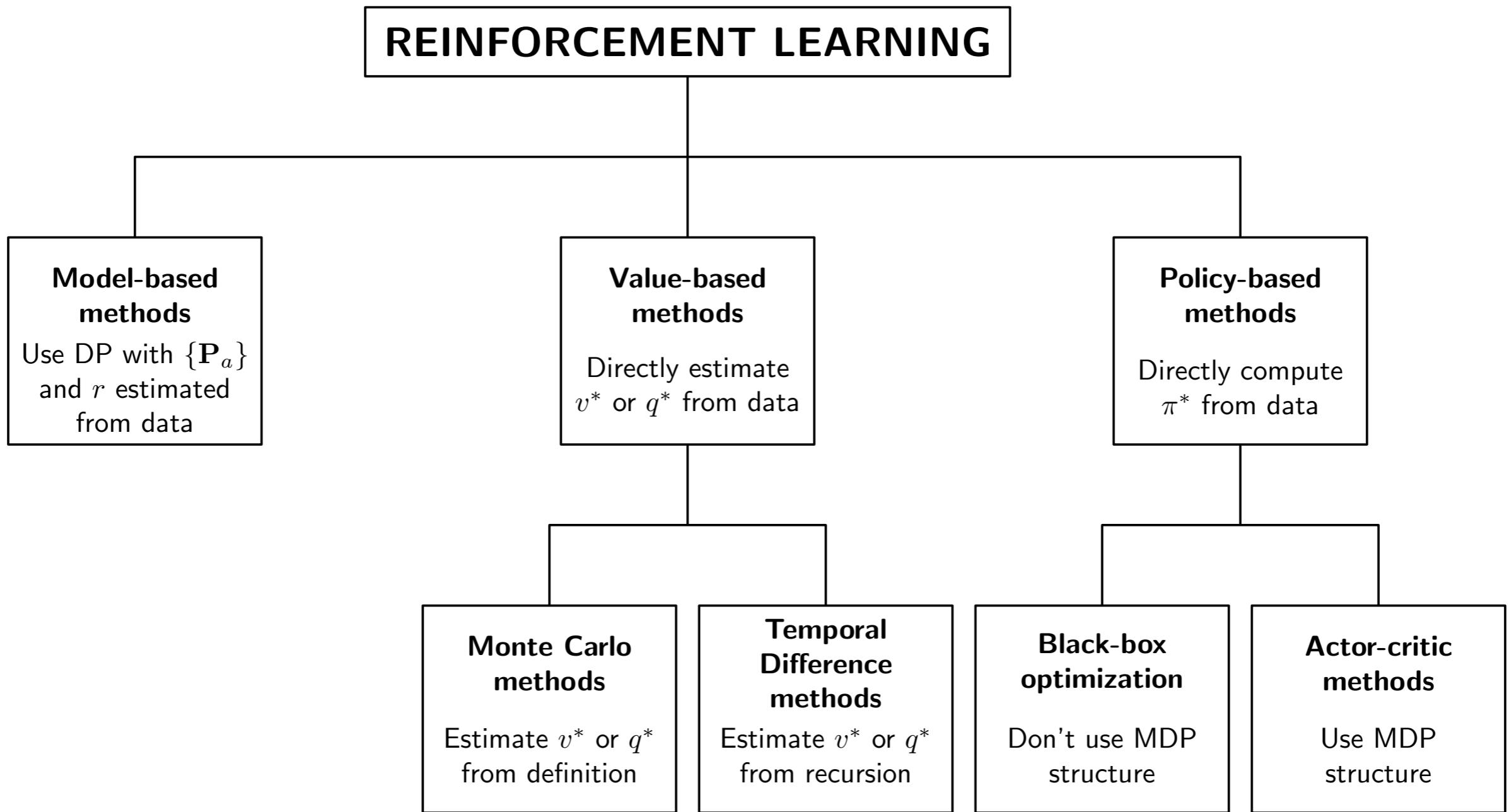


Taxonomy of RL methods

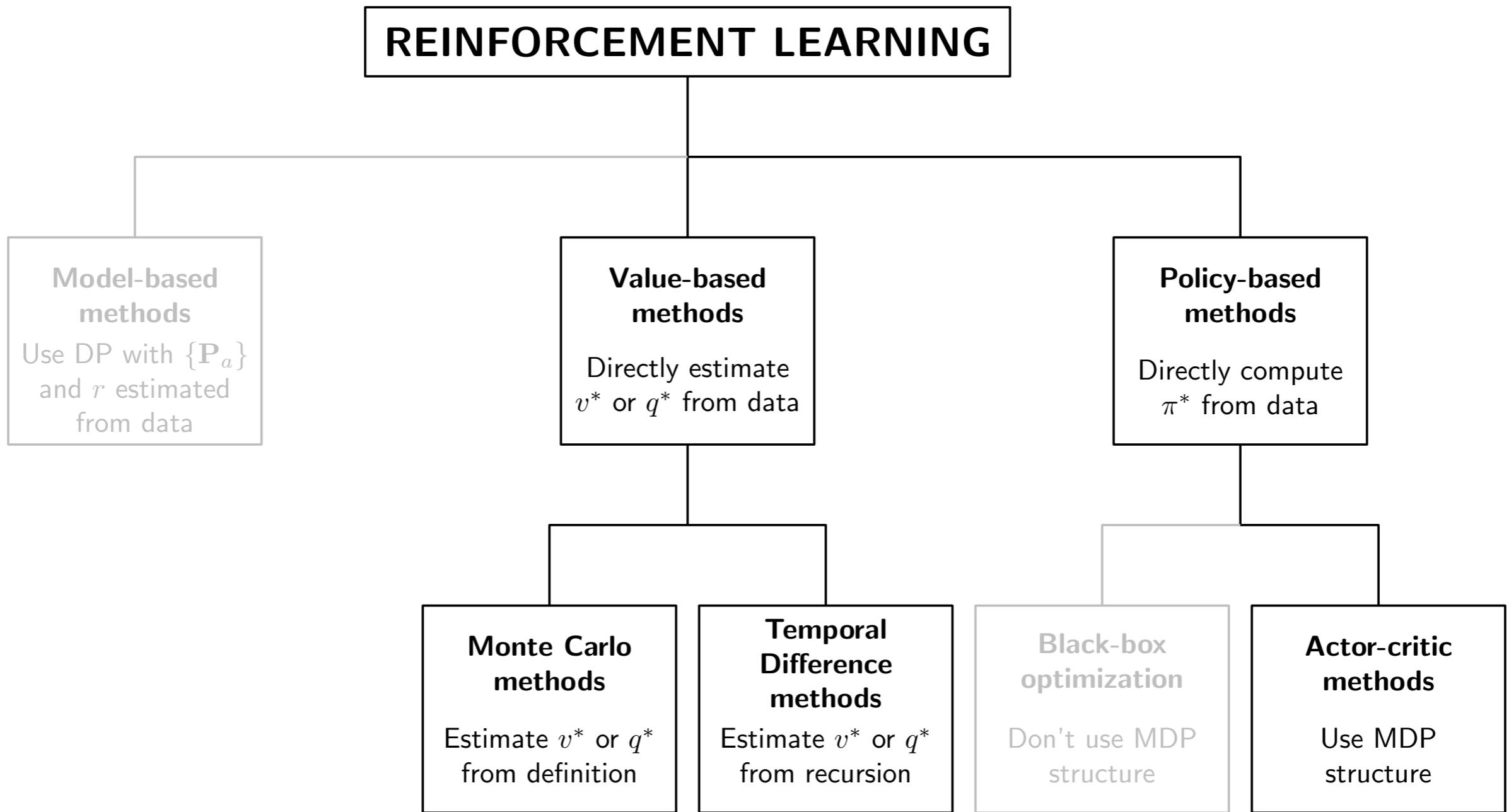
- **Policy-based methods:**



Taxonomy of RL methods



Taxonomy of RL methods



Monte Carlo approaches

The prediction problem

- We want to estimate v_π
- We are given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

obtained while following policy π

- We define the **return** as

$$G_0 = \sum_{t=0}^{T-1} \gamma^t r_t$$

Using the return

- From the definition of v_π ,

$$v_\pi(s_0) \approx \mathbb{E} [G_0]$$

- Then, given N trajectories with a common initial state s_0 , we can compute

$$\hat{v}(s_0) = \frac{1}{N} \sum_{n=1}^N G_{0,n}$$

or, incrementally,

$$\hat{v}(s_0) \leftarrow \hat{v}(s_0) + \frac{1}{N} (G_{0,N} - \hat{v}(s_0))$$



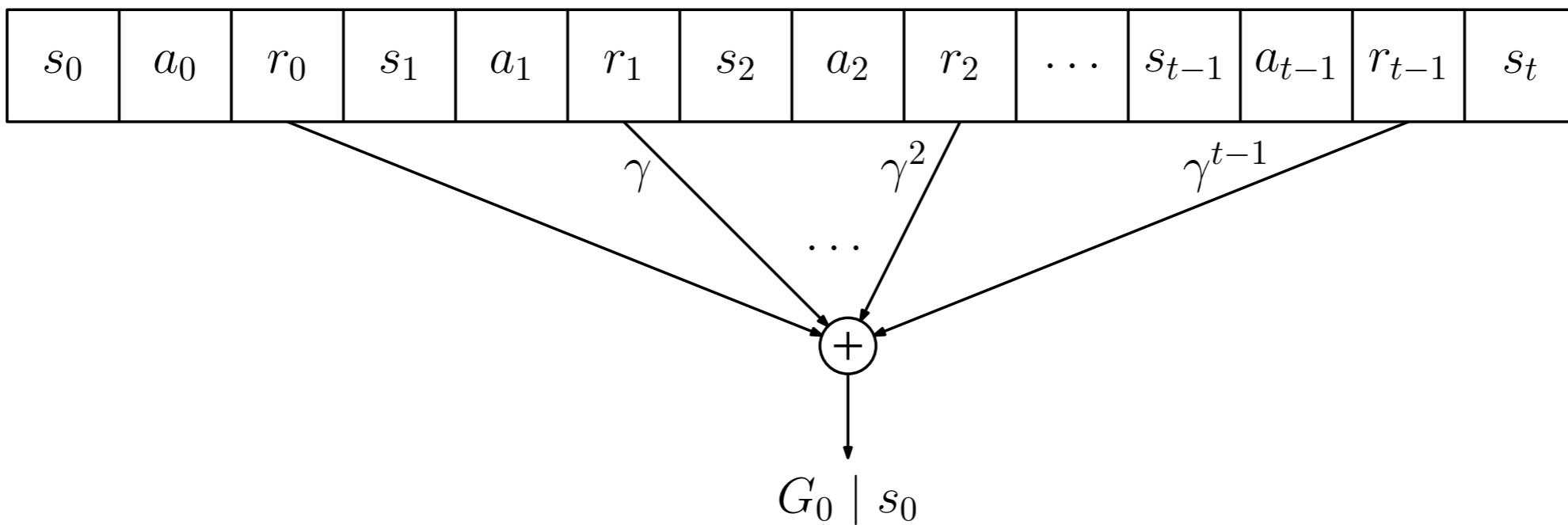
Return for trajectory N

Some considerations

- A trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

provides returns for multiple states

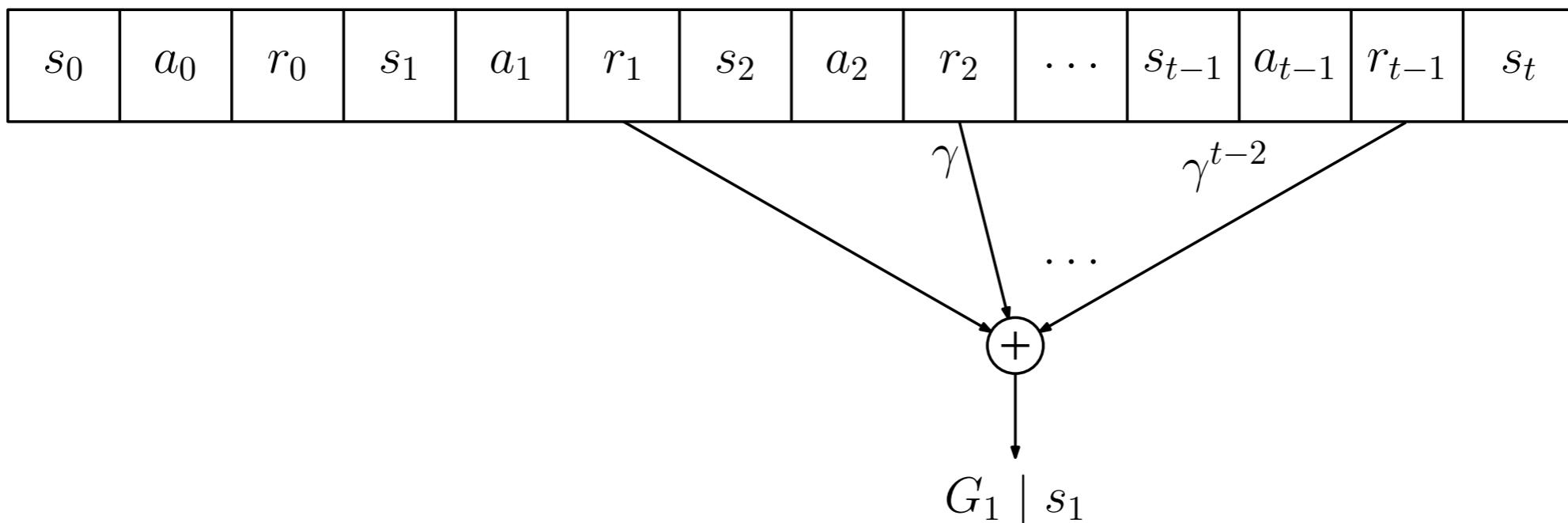


Some considerations

- A trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

provides returns for multiple states



Some considerations

- A trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

provides returns for multiple states

- Trajectories should visit all states a large number of times

The control problem

- We want to estimate q^*
- We are given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

obtained by selecting a random action a_0 and following a policy $\pi^{(0)}$ thereafter

Using the return

- From the definition of q_π ,

$$q_\pi(s_0, a_0) \approx \mathbb{E}[G_0]$$

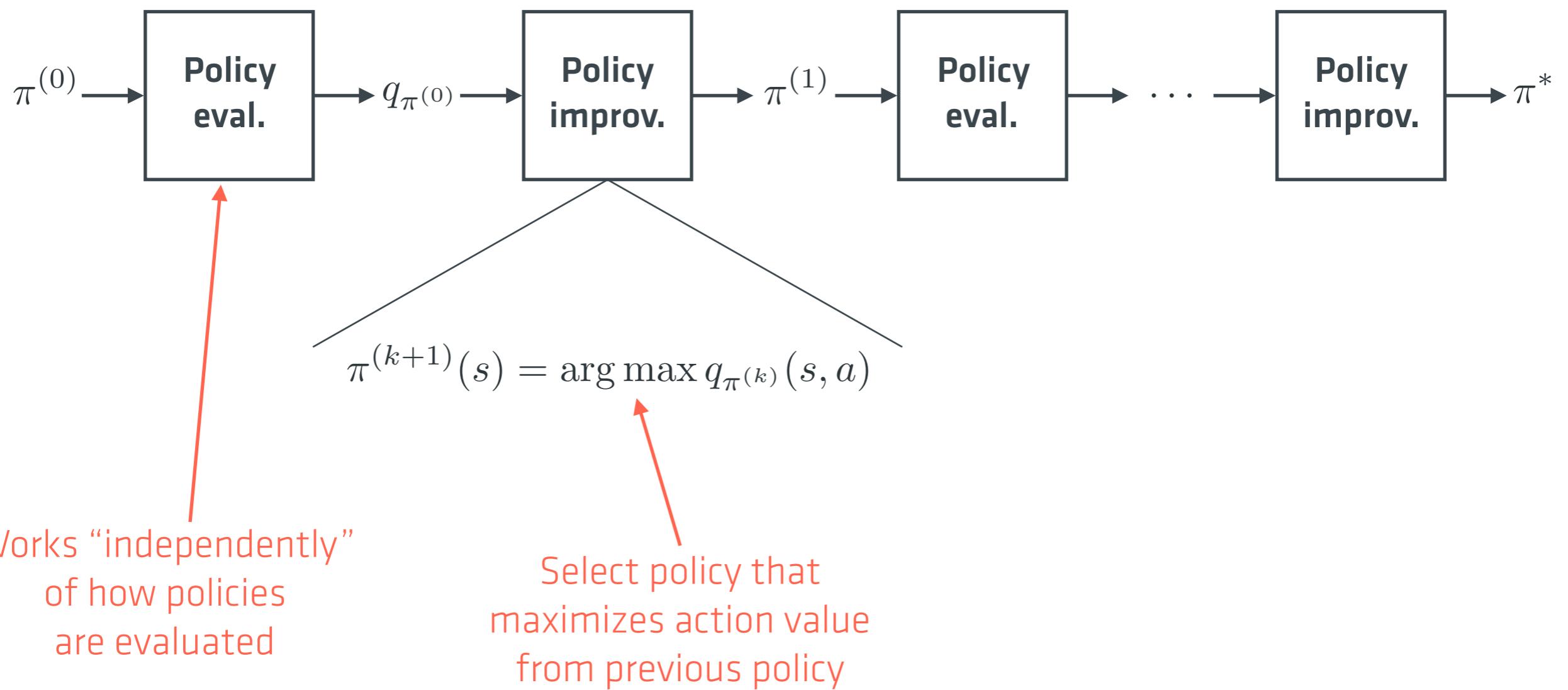
- Then, given N trajectories with a common initial state s_0 and initial action a_0 , we can compute

$$\hat{q}_\pi(s_0, a_0) = \frac{1}{N} \sum_{n=1}^N G_{0,n}$$

or, incrementally,

$$\hat{q}(s_0, a_0) \leftarrow \hat{q}(s_0, a_0) + \frac{1}{N} (G_{0,N} - \hat{q}(s_0, a_0))$$

Generalized policy iteration



Some considerations

- To estimate the Q-values for all state-action pairs, we need a large number of trajectories starting in each state-action pair
- To compute the optimal Q-values,
 - Start with arbitrary policy $\pi^{(0)}$ and set $k = 0$
 - Generate multiple trajectories, and estimate $q_{\pi^{(k)}}$
 - Compute policy
 - Improved policy**
 - $$\pi^{(k+1)}(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} q_{\pi^{(k)}}(s, a), \forall s$$
 - Set $k = k + 1$ and repeat

Deep Reinforcement Learning

Francisco S. Melo

Deep Structured Learning Course
29/11/2019

Outline of the lecture

- **Part I: RL Primer**

- The RL Problem
- Markov Decision Process - A Model for RL Problems
- Optimality & Dynamic Programming
- Monte Carlo Approaches
- Temporal Difference Learning
- The Policy Gradient Theorem

Outline of the lecture

- Part II: Deep RL
 - From RL to Deep RL
 - DQN
 - Deep advantage actor-critic methods
 - Trust region methods

Temporal difference learning

The prediction problem

- We want to estimate v_π
- We are given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

obtained while following policy π

The prediction problem

- We know that

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v_\pi(s') \right]$$

or, equivalently,

$$v_\pi(s) = \boxed{\mathbb{E}_{A_t \sim \pi(S_t)} [R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s]}$$

Expectation

The prediction problem

- We know that

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v_\pi(s') \right]$$

or, equivalently,

$$v_\pi(s) = \mathbb{E}_{A_t \sim \pi(S_t)} [R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

- The value function v_π can be computed iteratively via value iteration using the update

$$v^{(k+1)}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v^{(k)}(s') \right]$$

The prediction problem

- We know that

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) v_\pi(s') \right]$$

or, equivalently,

$$v_\pi(s) = \mathbb{E}_{A_t \sim \pi(S_t)} [R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

- The value function v_π can be computed iteratively via value iteration using the update

$$v^{(k+1)}(s) \leftarrow \mathbb{E}_{A_t \sim \pi(S_t)} [R_t + \gamma v^{(k)}(S_{t+1}) \mid S_t = s]$$

The prediction problem

- We can approximate the update

$$v^{(k+1)}(s) \leftarrow \mathbb{E}_{A_t \sim \pi(S_t)} [R_t + \gamma v^{(k)}(S_{t+1}) \mid S_t = s]$$

from samples $\{(s, r_n, s'_n)\}$ as

$$v^{(k+1)}(s) \leftarrow \frac{1}{N} \sum_{n=1}^N (r_n + \gamma v^{(k)}(s'_n))$$

or, incrementally,

$$v^{(k+1)}(s) \leftarrow v^{(k)}(s) + \frac{1}{N} (r_n + \gamma v^{(k)}(s'_n) - v^{(k)}(s))$$

Let's turn this into a proper algorithm

TD(0)

- Given a (potentially infinite) trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t, \dots\}$$

generated using policy π , and given an initial estimate $v^{(0)}$ for v_π , TD(0) performs, at each step t , the update

$$v^{(t+1)}(s_t) \leftarrow v^{(t)}(s_t) + \alpha_t (r_t + \gamma v^{(t)}(s_{t+1}) - v^{(t)}(s_t))$$

New estimate
(only updates component associated with current state s_t)
Old estimate
Step size
Temporal difference

TD(0)

- Given a (potentially infinite) trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t, \dots\}$$

generated using policy π , and given an initial estimate $v^{(0)}$ for v_π , TD(0) performs, at each step t , the update

$$v^{(t+1)}(s_t) \leftarrow v^{(t)}(s_t) + \alpha_t(r_t + \gamma v^{(t)}(s_{t+1}) - v^{(t)}(s_t))$$



Compare with what we had

$$v^{(k+1)}(s) \leftarrow v^{(k)}(s) + \frac{1}{N}(r_n + \gamma v^{(k)}(s'_n) - v^{(k)}(s))$$

The control problem

- We want to estimate q^*
- We start with the idea used in MC methods (compute q_π , improve π , repeat)
- We are given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

obtained while following some initial policy π

The control problem

- Repeating the same reasoning,

$$q_\pi(s, a) = \mathbb{E}_{A_{t+1} \sim \pi(S_{t+1})} [R_t + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

leading to the update

$$q^{(k+1)}(s, a) \leftarrow \mathbb{E}_{A_{t+1} \sim \pi(S_{t+1})} [R_t + \gamma q^{(k)}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

The control problem

- Then, given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

generated using a policy π , and given an initial estimate $q^{(0)}$ for q_π , update

$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t(r_t + \gamma q^{(t)}(s_{t+1}, a_{t+1}) - q^{(t)}(s_t, a_t))$$

- After some iterations, compute a new policy

$$\pi(s) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} q^{(t)}(s, a)$$

SARSA

- This approach runs the following cycle:
 - Start with a policy
 - Evaluate it, computing its associated Q-function
 - Update the policy
 - Repeat
- Each update to $q^{(t)}$ uses a sample $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$
- The algorithm is thus named SARSA

Can we learn q^* directly?

The control problem

- Let us again repeat the same reasoning

$$q^*(s, a) = \mathbb{E} \left[R_t + \gamma \max_{a \in \mathcal{A}} q^*(S_{t+1}, a) \mid S_t = s, A_t = a \right]$$

we get the update

$$q^{(k+1)}(s, a) \leftarrow \mathbb{E} \left[R_t + \gamma \max_{a \in \mathcal{A}} q^{(k)}(S_{t+1}, a) \mid S_t = s, A_t = a \right]$$

Q-learning

- Then, given a (potentially infinite) trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t, \dots\}$$

generated using an arbitrary policy π , and given an initial estimate $q^{(0)}$ for q^* , update

$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t(r_t + \gamma \max_{a \in \mathcal{A}} q^{(t)}(s_{t+1}, a) - q^{(t)}(s_t, a_t))$$

Summarizing...

- TD(0) is used to compute the value function for a given policy
- It relies on the update

$$v^{(t+1)}(s_t) \leftarrow v^{(t)}(s_t) + \alpha_t(r_t + \gamma v^{(t)}(s_{t+1}) - v^{(t)}(s_t))$$

Summarizing...

- SARSA and Q-learning are used to compute the optimal Q-function
- SARSA relies on the update

$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t (r_t + \gamma q^{(t)}(s_{t+1}, a_{t+1}) - q^{(t)}(s_t, a_t))$$

- SARSA learns the Q-function for the policy used to obtain the samples

☞ On-policy learning

- In order to compute the optimal policy, it must slowly adjust the policy used to obtain the samples

Summarizing...

- Q-learning relies on the update

$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t(r_t + \gamma \max_{a \in \mathcal{A}} q^{(t)}(s_{t+1}, a) - q^{(t)}(s_t, a_t))$$

- Q-learning learns the optimal Q-function, independently of the policy used to obtain the samples

☞ Off-policy learning

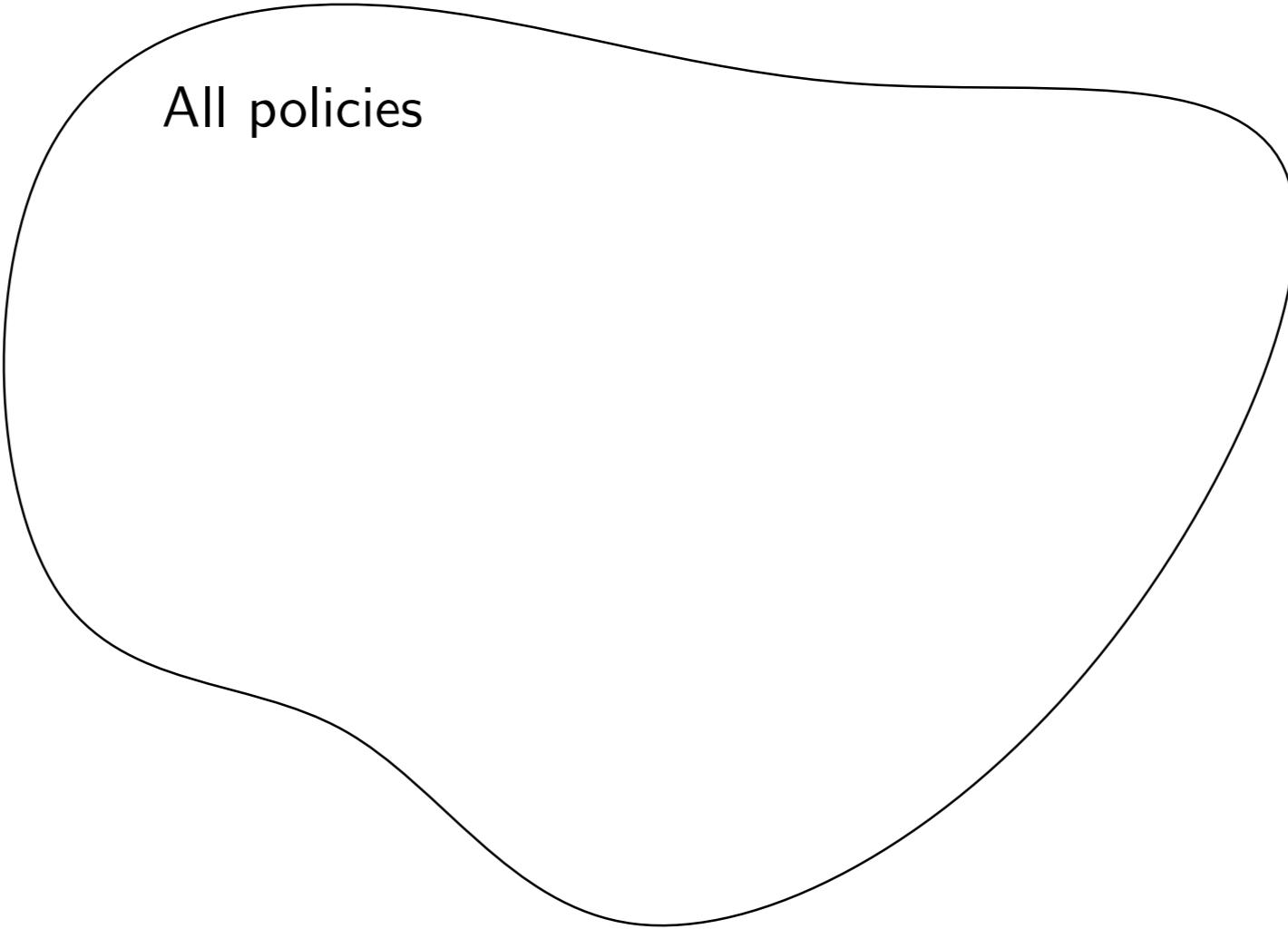
The policy gradient theorem

Policy-based methods

- The goal is to compute π^* directly
- We depart from a parameterized family of policies, π_θ

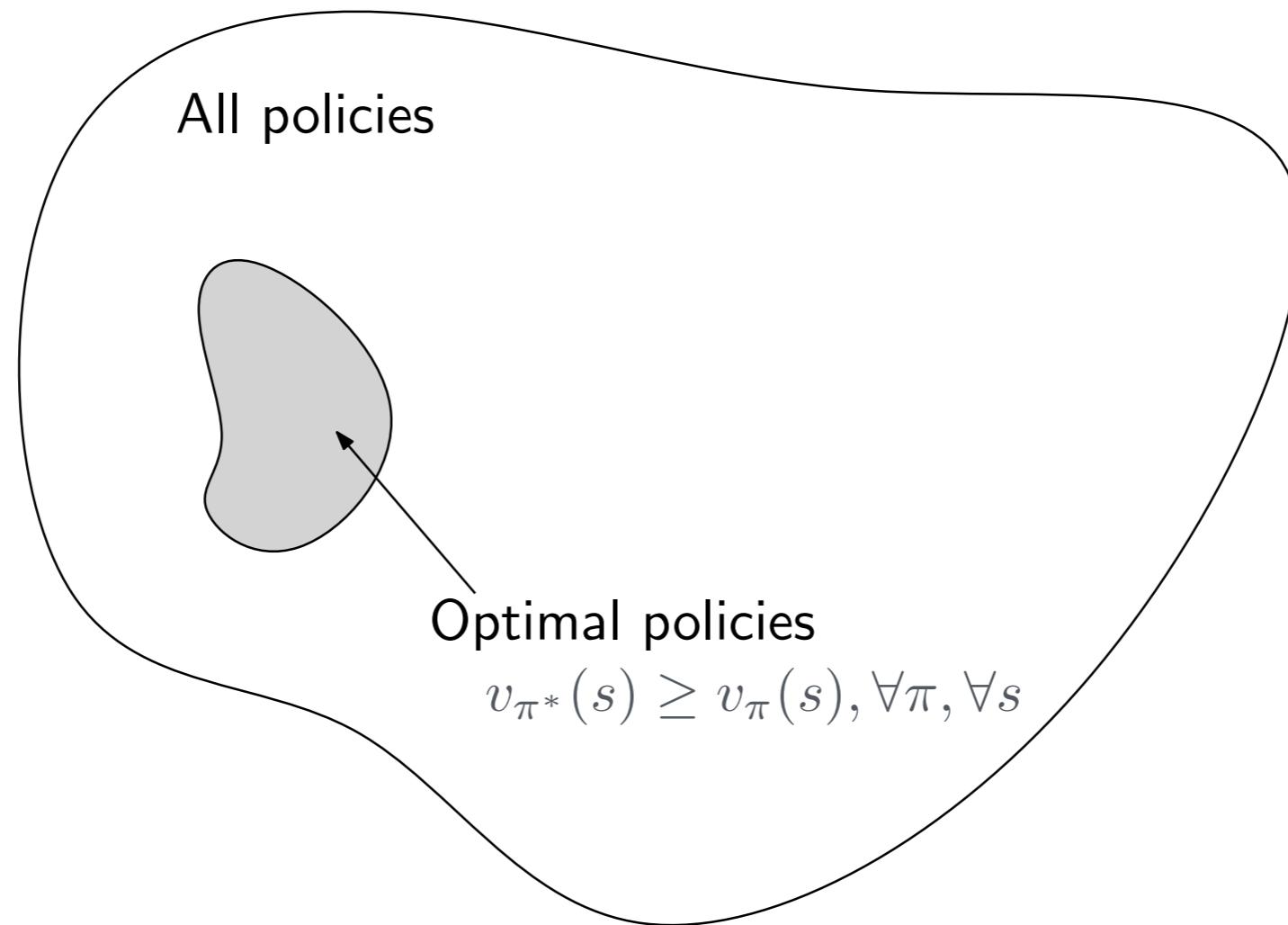
... however...

Policy-based methods

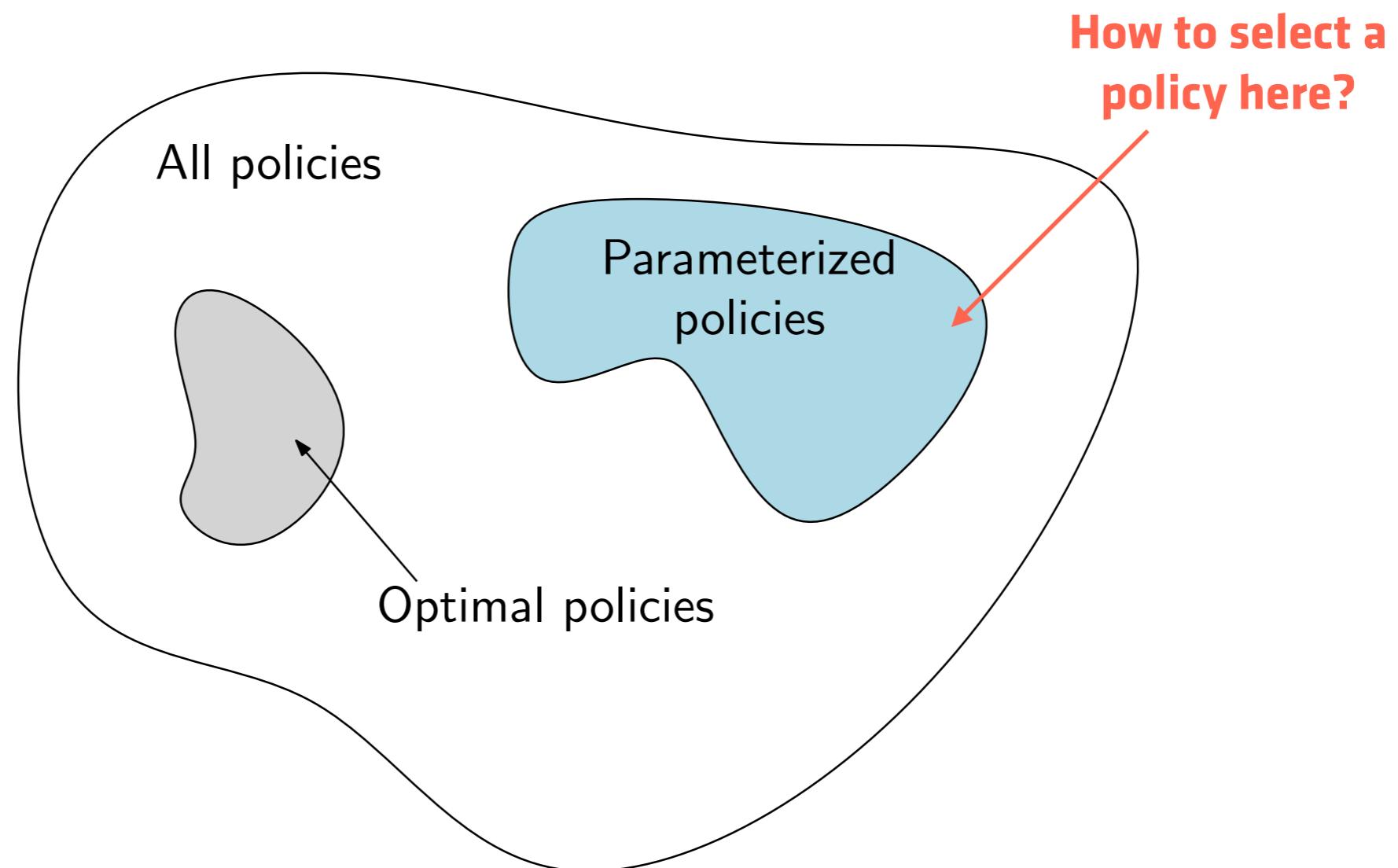


All policies

Policy-based methods



Policy-based methods



Revisiting optimality criterion

- When considering the set of all policies, state-wise optimization is **possible**
- When considering a restricted set of policies, state-wise optimization **may not be possible**

Revisiting optimality criterion

- Recall that our goal is to maximize

$$J(\{R_t, t = 0, \dots, \}) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right]$$

- We consider that the initial state of the MDP follows some **initial distribution μ**
- To explicitly indicate the dependence of J on the **initial distribution μ** and the **policy π** used to generate $\{R_t, t = 1, \dots\}$, we write

$$J(\pi; \mu) \triangleq \mathbb{E}_{\pi} \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 \sim \mu \right]$$

Interesting relations

- We have that
 - $v_\pi(s) = J(\pi; \mu)$ when $\mu(s') = \mathbb{I}(s' = s)$
 - Conversely, for an arbitrary distribution μ ,

$$J(\pi; \mu) = \sum_{s \in \mathcal{S}} \mu(s) v_\pi(s)$$

RL using gradient ascent

- We can now optimize J with respect to the parameters of the policy
- Using gradient ascent, we get an algorithm

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\pi_{\theta}; \mu)$$



**Methods based on this idea
are globally called
“policy-gradient methods”**

Policy gradient

- We now compute the policy gradient

$$\begin{aligned}\nabla_{\theta} J(\pi_{\theta}; \mu) &= \nabla_{\theta} \sum_{s \in \mathcal{S}} \mu(s) v_{\pi_{\theta}}(s) \\ &= \sum_{s \in \mathcal{S}} \mu(s) \boxed{\nabla_{\theta} v_{\pi_{\theta}}(s)}\end{aligned}$$

Let us consider
this term alone

Policy gradient

- Since

$$v_{\pi_\theta}(s) = \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) q_{\pi_\theta}(s, a)$$

it holds that

$$\nabla_\theta v_{\pi_\theta}(s) = \sum_{a \in \mathcal{A}} [\nabla_\theta \pi_\theta(a \mid s) q_{\pi_\theta}(s, a) + \pi_\theta(a \mid s) \boxed{\nabla_\theta q_{\pi_\theta}(s, a)}]$$

We now look
at this term

Policy gradient

- Since

$$q_{\pi_\theta}(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) v_{\pi_\theta}(s')$$

it holds that

$$\nabla_\theta q_{\pi_\theta}(s, a) = \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) \nabla_\theta v_{\pi_\theta}(s')$$

Policy gradient

- Putting everything together,

$$\nabla_{\theta} v_{\pi_{\theta}}(s) = \sum_{a \in \mathcal{A}} \left[\nabla_{\theta} \pi_{\theta}(a | s) q_{\pi_{\theta}}(s, a) + \gamma \pi_{\theta}(a | s) \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right]$$

$$= \sum_{a \in \mathcal{A}} \pi_{\theta}(a | s) \left[\frac{\nabla_{\theta} \pi_{\theta}(a | s)}{\pi_{\theta}(a | s)} q_{\pi_{\theta}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' | s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right]$$

Factoring this out

This is just
 $\nabla_{\theta} \log \pi_{\theta}(a | s)$

Policy gradient

- Putting everything together,

$$\begin{aligned}\nabla_{\theta} v_{\pi_{\theta}}(s) &= \sum_{a \in \mathcal{A}} \left[\nabla_{\theta} \pi_{\theta}(a \mid s) q_{\pi_{\theta}}(s, a) + \gamma \pi_{\theta}(a \mid s) \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right] \\ &= \sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid s) \left[\nabla_{\theta} \log \pi_{\theta}(a \mid s) q_{\pi_{\theta}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbf{P}(s' \mid s, a) \nabla_{\theta} v_{\pi_{\theta}}(s') \right]\end{aligned}$$

- Recursive relation reminiscent of that for v_{π}



**Plays the role
of “reward”**

Policy gradient

- Unfolding the recursion finally yields

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \sum_{s \in \mathcal{S}} \mu_{\theta}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid s) \nabla_{\theta} \log \pi_{\theta}(a \mid s) q_{\pi_{\theta}}(s, a)$$

or, equivalently,

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A \mid S) q_{\pi_{\theta}}(S, A)]$$

- The distribution μ_{θ} translates the “discounted visitation frequency” under π_{θ}
- Can be sampled by sampled the MDP while following π_{θ}

REINFORCE

- The gradient is just

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) q_{\pi_{\theta}}(S, A)]$$

- Given a trajectory obtained from π_{θ} and with initial state sampled from μ_{θ} ,

$$\nabla_{\theta} J(\pi_{\theta}; \mu) \approx \sum_{t=0}^T \gamma^t G_t \log \pi_{\theta}(a_t | s_t)$$



Estimate of
 $q_{\pi}(s_t, a_t)$

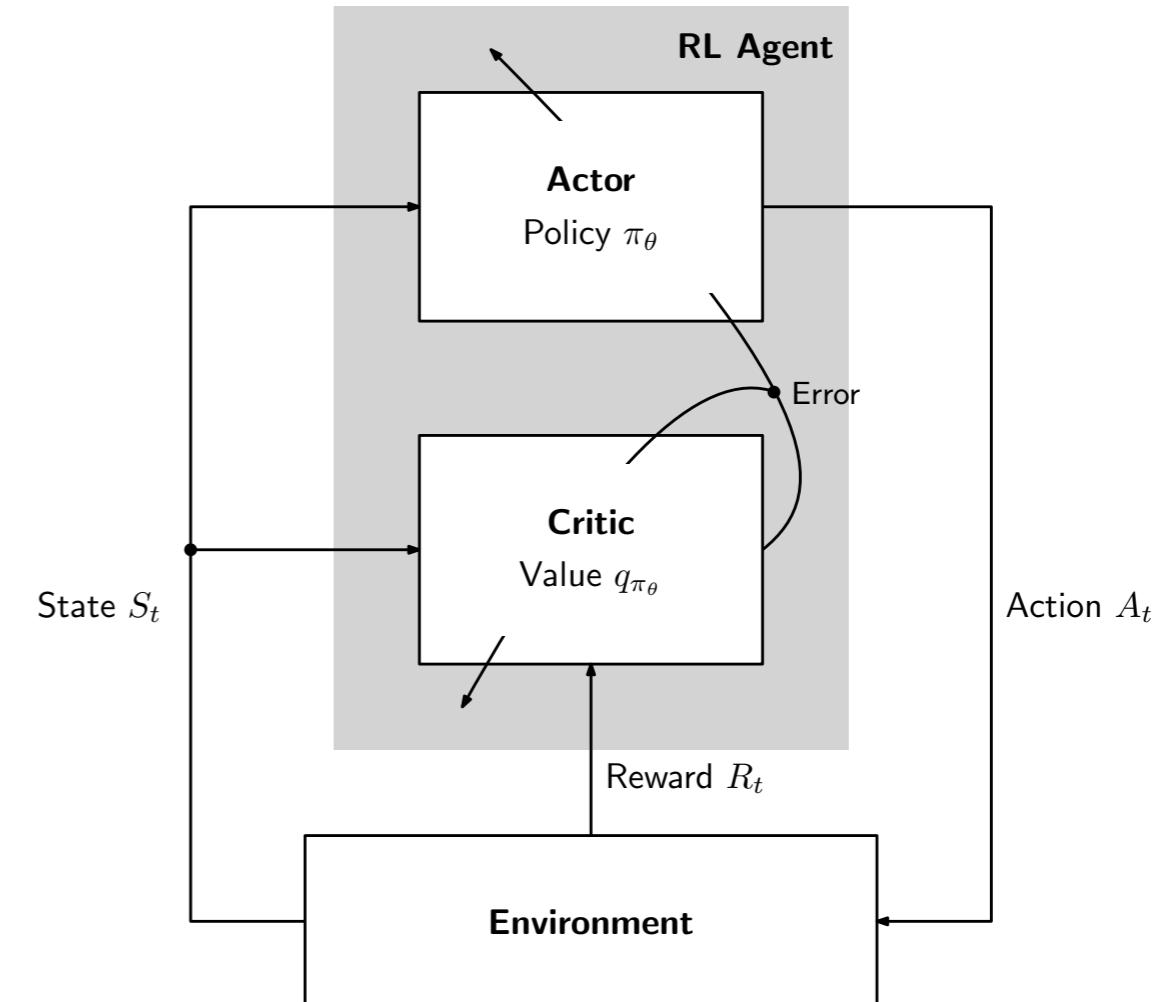
Actor-critic architecture

- To compute the gradient, we require an estimate of the Q-values
- REINFORCE uses a simple Monte Carlo approach to build such estimate
- However, other approaches can be used (e.g., temporal-difference learning)

Actor-critic architecture

- The RL algorithm comprises two components:
 - An **actor**, responsible for executing the policy π_θ
 - A **critic**, responsible for evaluating the policy (computing q_π)

**Actor-critic
architecture**



TD-based actor-critic

- For example, we can have an actor-critic based on TD-learning:
 - Given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_t, \dots\}$$

- Update the Q-value estimates as

$$q^{(t+1)}(s_t, a_t) = q^{(t)}(s_t, a_t) + \alpha_t(r_t + \gamma q^{(t)}(s_{t+1}, a_{t+1}) - q^{(t)}(s_t, a_t))$$

- Update gradient term

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \beta_t \gamma^t q^{(t+1)}(s_t, a_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(s_t, a_t)$$

Considerations

- PG/AC architectures are convenient with **function approximation**
 - Gradient does not depend on q_π but on a projection thereof
 - Variations of the gradient (e.g., **natural gradient**) can also be used
 - Discount is cumbersome to deal with
 - Many PG/AC applications instead adopt the **average per-step reward**
 - **Fully incremental approaches** suffer from high variance and are seldom used

Adding a baseline

- Consider once again the gradient expression

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi_{\theta}(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) q_{\pi_{\theta}}(S, A)]$$

- Gradient estimated from **samples**
- Estimates plagued by **high variance** (sensitivity to the particular samples)

Adding a baseline

- Result from theory of Monte Carlo integration:
 - Use of a **baseline** can often improve variance of sample-based estimates

$$\mathbb{E} [f(X)] \approx \frac{1}{N} \sum_{n=1}^N f(x_n)$$

$$\mathbb{E} [f(X) - g(X)] \approx \frac{1}{N} \sum_{n=1}^N (f(x_n) - g(x_n)) \longrightarrow \text{Less variance}$$

Baseline
($\mathbb{E} [g(X)]$ known)



Adding a baseline

- Consider an arbitrary function

$$b : \mathcal{S} \rightarrow \mathbb{R}$$

- Then,

$$\sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) b(s) = ?$$

Adding a baseline

- Consider an arbitrary function

$$b : \mathcal{S} \rightarrow \mathbb{R}$$

- Then,

$$\sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) b(s) = \nabla_\theta \left[\sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) \right] b(s) = 0$$

Adding a baseline

- But then

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) q_{\pi_{\theta}}(S, A) - \nabla_{\theta} \log \pi_{\theta}(A | S) b(S)]$$

or, equivalently,

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) (q_{\pi_{\theta}}(S, A) - b(S))]$$

Best baseline:

$$v_{\pi_{\theta}}(S)$$

Adding a baseline

- But then

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) q_{\pi_{\theta}}(S, A) - \nabla_{\theta} \log \pi_{\theta}(A | S) b(S)]$$

or, equivalently,

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) (q_{\pi_{\theta}}(S, A) - v_{\pi_{\theta}}(S))]$$

Advantage
 $\text{adv}_{\pi}(S, A)$



Adding a baseline

- But then

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) q_{\pi_{\theta}}(S, A) - \nabla_{\theta} \log \pi_{\theta}(A | S) b(S)]$$

or, equivalently,

$$\nabla_{\theta} J(\pi_{\theta}; \mu) = \mathbb{E}_{S \sim \mu_{\theta}, A \sim \pi(\cdot | S)} [\nabla_{\theta} \log \pi_{\theta}(A | S) \text{adv}_{\pi_{\theta}}(S, A)]$$

☞ This is the underlying form of most current AC algorithms

Outline of the lecture

- **Part I: RL Primer**

- The RL Problem
- Markov Decision Process - A Model for RL Problems
- Optimality & Dynamic Programming
- Monte Carlo Approaches
- Temporal Difference Learning
- The Policy Gradient Theorem

Outline of the lecture

- **Part II: Deep RL**
 - From RL to Deep RL
 - DQN
 - Deep advantage actor-critic methods
 - Trust region methods

RL in large domains

- **Plan:**
 - Revisit **temporal difference learning** in large domains
 - Revisit **policy-gradient methods** in large domains

Temporal difference learning revisited

TDL in large domains

- Temporal difference learning methods require **explicit updates**:

$$v^{(t+1)}(s_t) \leftarrow v^{(t)}(s_t) + \alpha_t(r_t + \gamma v^{(t)}(s_{t+1}) - v^{(t)}(s_t))$$
$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t(r_t + \gamma \max_{a \in \mathcal{A}} q^{(t)}(s_{t+1}, a) - q^{(t)}(s_t, a_t))$$

Component
 (s_t, a_t) **is explicitly updated**

Component
 s_t **is explicitly updated**

TDL in large domains

- For large domains, **function approximation** is necessary
 - We can no longer compute v_π or q^* exactly
 - Instead, we consider parameterized families of functions

TDL in large domains

- Example: TD-learning with linear function approximation
 - We consider the family of functions of the form

$$v(s; \boldsymbol{w}) = \boldsymbol{w}^\top \phi(s)$$

where \boldsymbol{w} is a vector of parameters

- We update the parameters \boldsymbol{w} as

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \alpha_t \phi(s_t) (r_t + \gamma v(s_{t+1}; \boldsymbol{w}^{(t)}) - v(s_t; \boldsymbol{w}^{(t)}))$$

 Compare

$$v^{(t+1)}(s_t) \leftarrow v^{(t)}(s_t) + \alpha_t (r_t + \gamma v^{(t)}(s_t) - v^{(t)}(s_t))$$

TDL in large domains

- Another example: Q-learning with linear function approximation
 - We consider the family of functions of the form

$$q(s, a; \mathbf{w}) = \mathbf{w}^\top \phi(s, a)$$

where w is a vector of parameters

- We update the parameters \mathbf{w} as

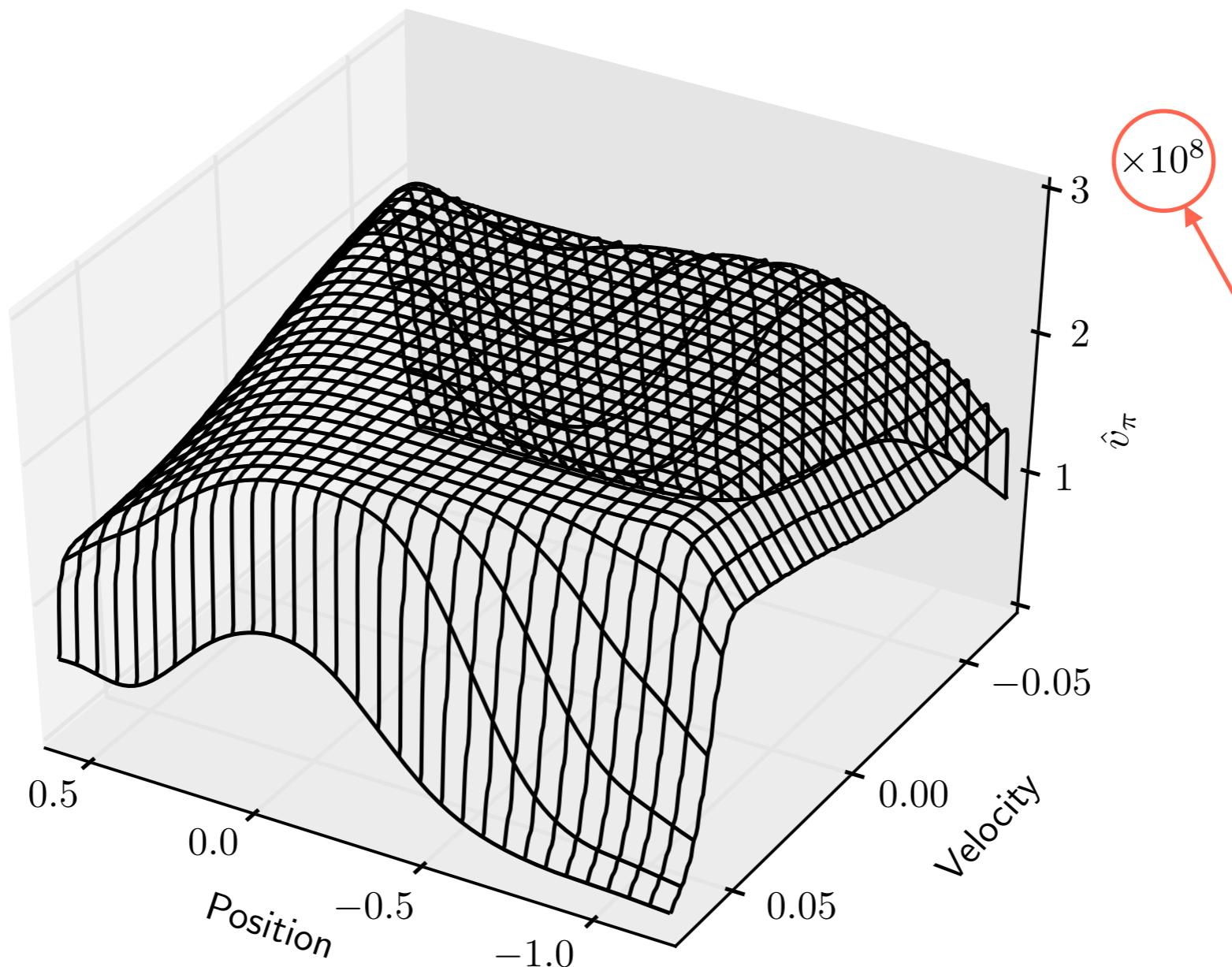
$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \alpha_t \phi(s_t, a_t) (r_t + \gamma \max_{a \in \mathcal{A}} q(s_{t+1}, a; \mathbf{w}^{(t)}) - q(s_t, a_t; \mathbf{w}^{(t)}))$$



$$q^{(t+1)}(s_t, a_t) \leftarrow q^{(t)}(s_t, a_t) + \alpha_t (r_t + \gamma \max_{a \in \mathcal{A}} q^{(t)}(s_{t+1}, a) - q^{(t)}(s_t, a_t))$$

The problem of function approximation

- Unfortunately, temporal-difference methods may **diverge** with function approximation



The problem of function approximation

- Issues with function approximation in RL:
 - Bootstrapping - the target is built from current estimate
 - Sample correlation - samples come from a trajectory

Given the previous difficulties, how can we
combine ANNs with RL?

Combining ANNs and RL

- We address directly the control problem
- Three ideas:
 - Create a **replay buffer** to avoid sample correlation
 - Use an auxiliary estimate for q^* (a **target network**) to avoid bootstrapping
 - Turn the trajectory data into supervised learning data

1. Build replay buffer

- Given a trajectory

$$\mathcal{T} = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

create a set of transitions (replay buffer)

$$\mathcal{T}' = \{(s_t, a_t, r_t, s_{t+1}), t = 0, \dots, T - 1\}$$



At training time, we
select random transitions
from the replay buffer



Goal: minimize
sample correlation

2. Build targets

- At training time, given a sample (s_t, a_t, r_t, s_{t+1}) from the replay buffer, build target

$$y_t = r_t + \max_{a \in \mathcal{A}} \hat{q}(s_{t+1}, a)$$

where \hat{q} is an estimate of q^*

**Auxiliary estimate
(target network)**

- We thus build a dataset

$$\mathcal{D} = \{(s_{t_k}, a_{t_k}, y_{t_k}), k = 1, \dots, K\}$$

3. Train

- The error associated with sample t_k is now

$$\varepsilon_k = (y_{t_k} - q(s_{t_k}, a_{t_k}; \mathbf{w}))^2$$

with gradient

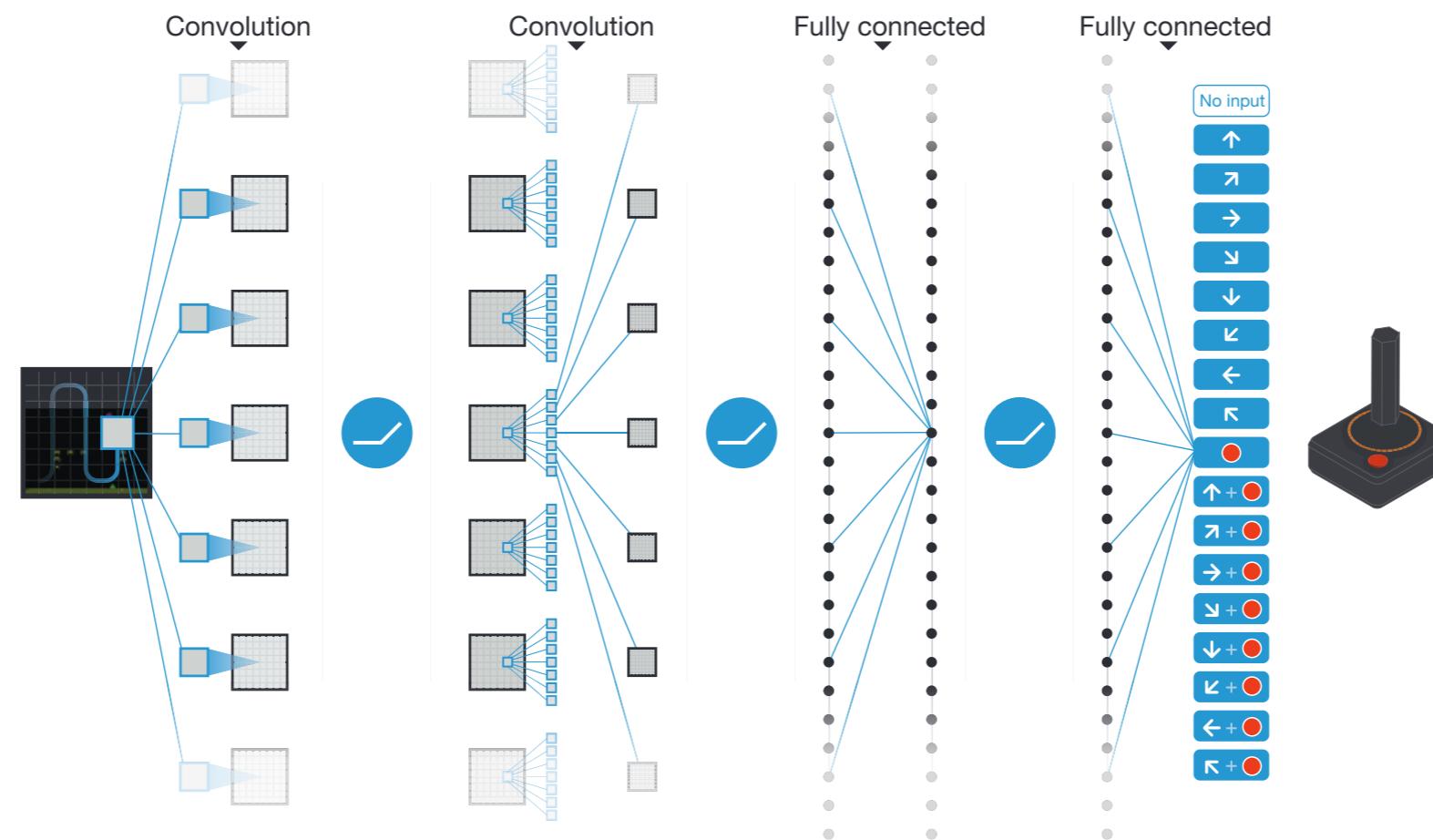
$$\nabla_{\mathbf{w}} \varepsilon_k = -2\nabla_{\mathbf{w}} q(s_{t_k}, a_{t_k}; \mathbf{w})(y_{t_k} - q(s_{t_k}, a_{t_k}; \mathbf{w}))$$

$$= -2\nabla_{\mathbf{w}} q(s_{t_k}, a_{t_k}; \mathbf{w})(r_{t_k} + \gamma \max_{a \in \mathcal{A}} \hat{q}(s_{t+1}, a) - q(s_{t_k}, a_{t_k}; \mathbf{w}))$$

Resembles
 Q-learning
 update
 

DQN

- The resulting approach is known as a **Deep Q-Network** (DQN)
- It was the approach used in the ATARI deep RL paper



DQN

- **Some considerations:**

- The DQN network takes the **state as input** and has **one output per action**
- The target network is a **copy** of the DQN, i.e.,
$$\hat{q}(s, a) = q(s, a; \boldsymbol{w}^-)$$
**“Old” parameters**
- It is updated every C steps with the weights of the main DQN

Variations: DDQN

- The targets in DQN are computed as

$$y_t = r_t + \max_{a \in \mathcal{A}} q(s_{t+1}, a; \mathbf{w}^-)$$

where the target network seeks to avoid bootstrapping

- We can further decouple:
 - ... the computation of the **maximizing action**; and
 - ... the **value** of the maximizing action.

Variations: DDQN

- The targets in double DQN (DDQN), the targets are computed as

$$y_t = r_t + q(s_{t+1}, \operatorname{argmax}_{a \in \mathcal{A}} q(s_{t+1}, a; \mathbf{w}); \mathbf{w}^-)$$

**Target network is used
to compute the
maximizing value**

**Original network is used
to compute the
maximizing action**

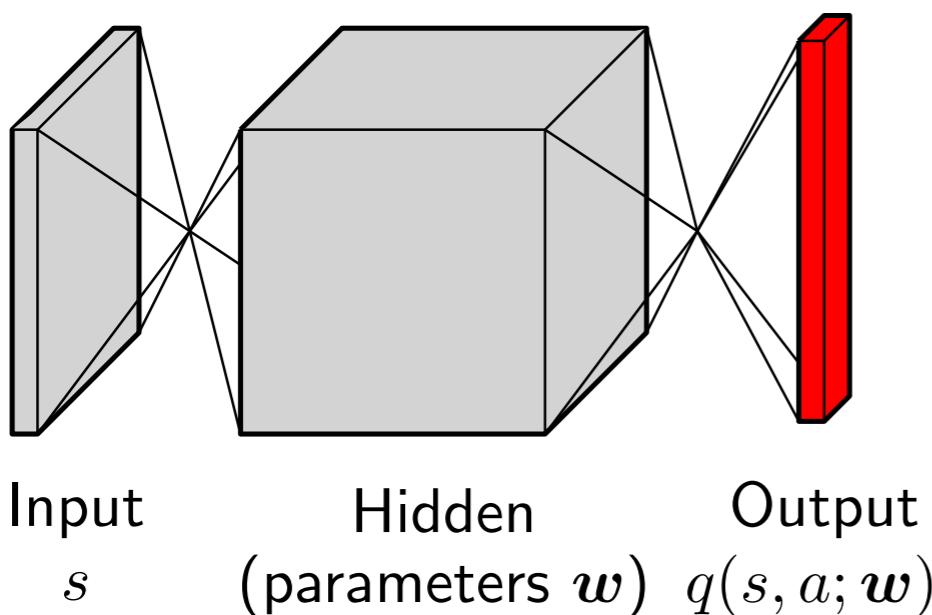
More variations

- Prioritized replay:
 - Transitions are sampled from the replay memory with a probability that increases with the associated error:

$$\varepsilon_k = (y_{t_k} - q(s_{t_k}, a_{t_k}; \mathbf{w}))^2$$

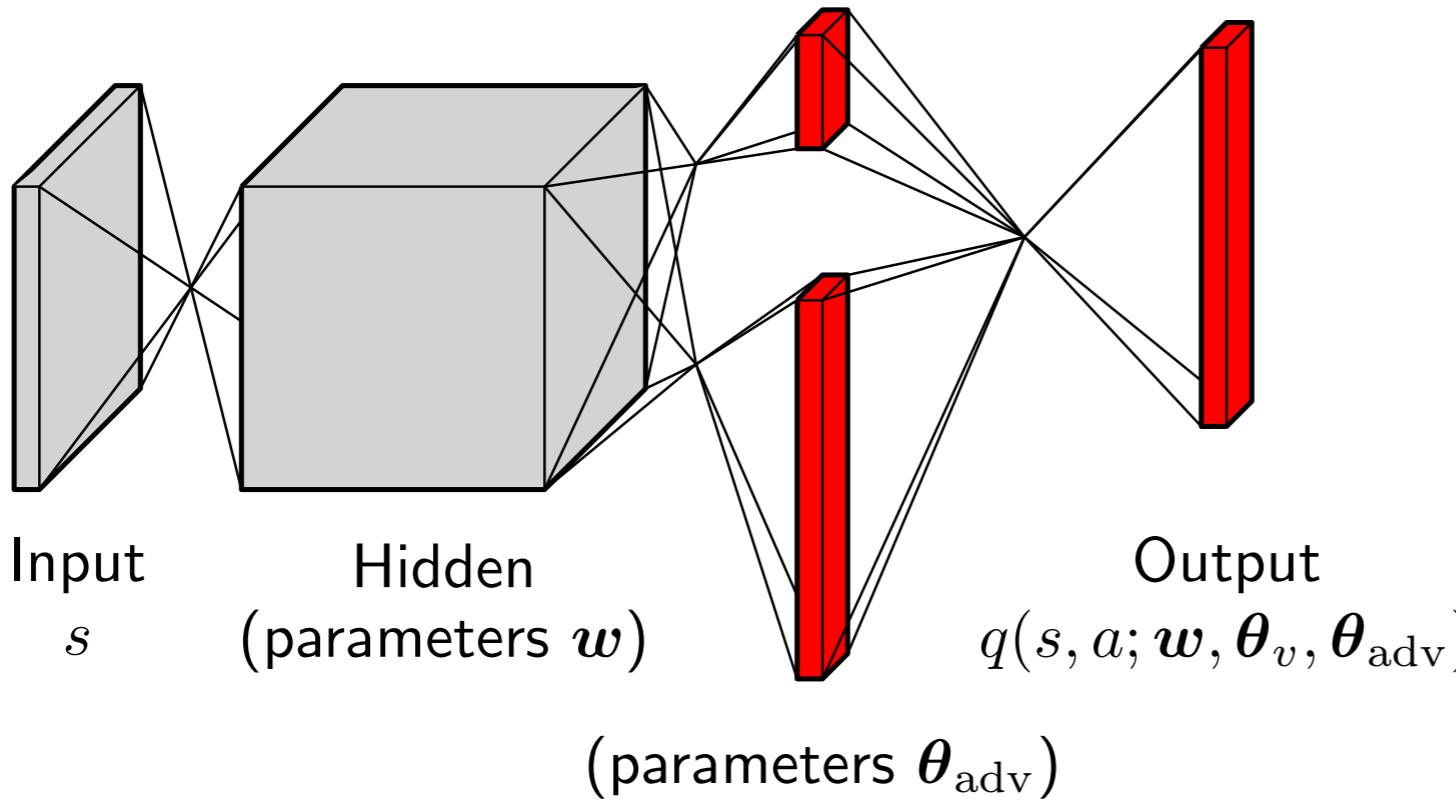
More variations

- Dueling network:
 - Instead of the “standard” DQN architecture



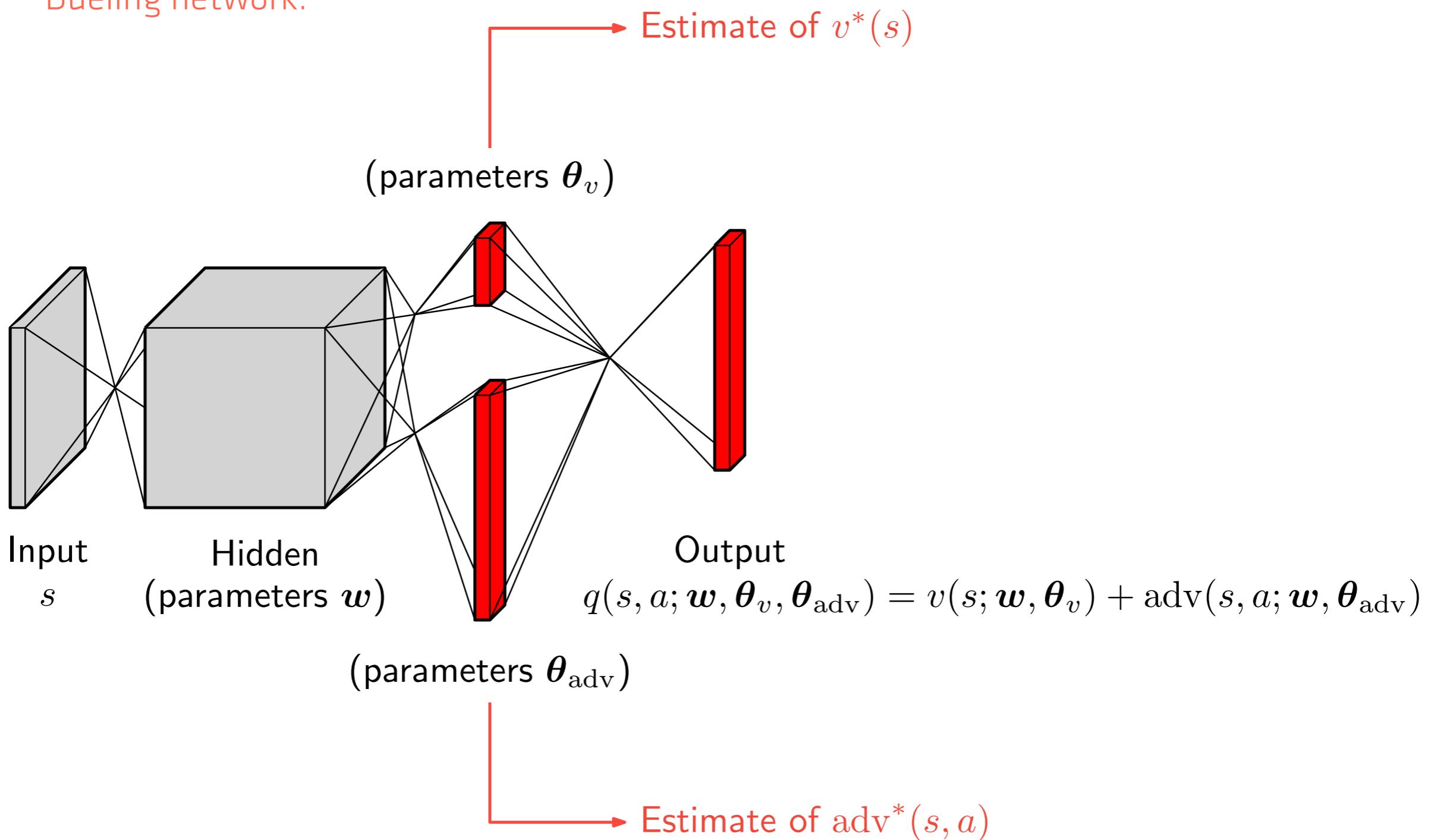
More variations

- Dueling network:
 - Instead of the “standard” DQN architecture, dueling networks propose (parameters θ_v)



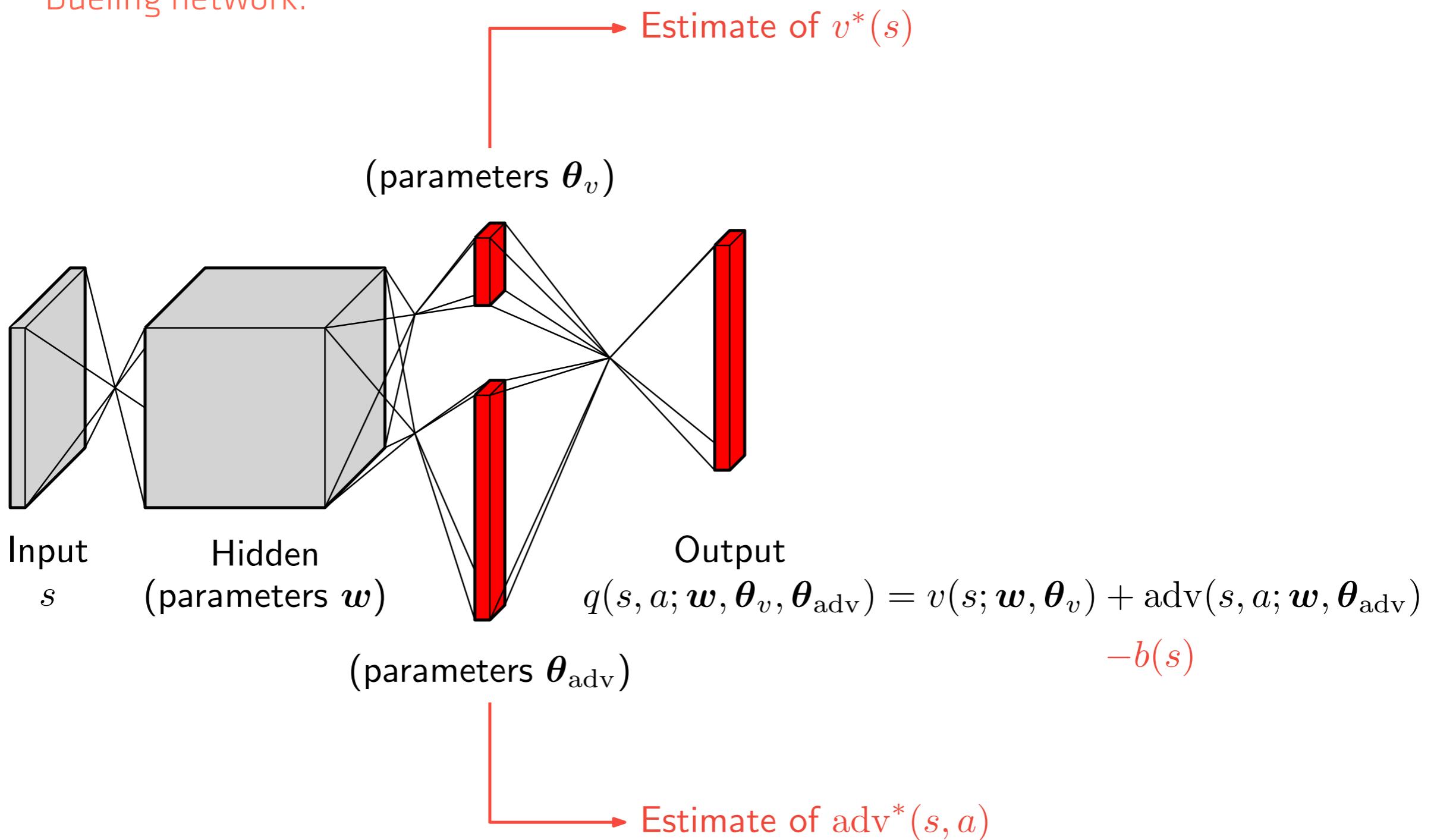
More variations

- Dueling network:



More variations

- Dueling network:



Considerations

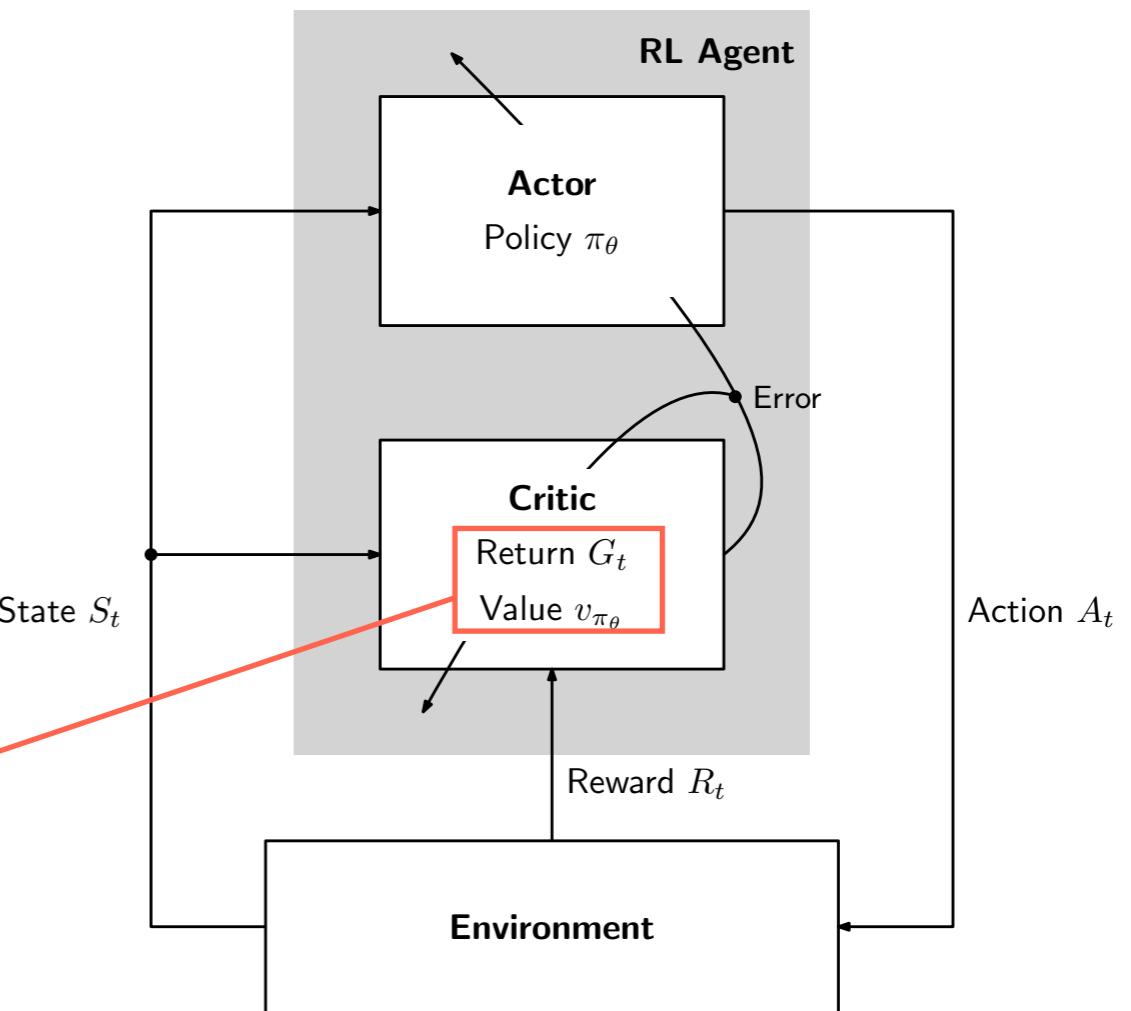
- Different variations offer different advantages:
 - DDQN - more stable learning than DQN
 - Prioritized replay - better use of memory (faster learning)
 - Dueling DQN - better performance, particularly in domains where actions only relevant in some states
- Different variations are mostly orthogonal, and can be combined

Policy gradient methods revisited

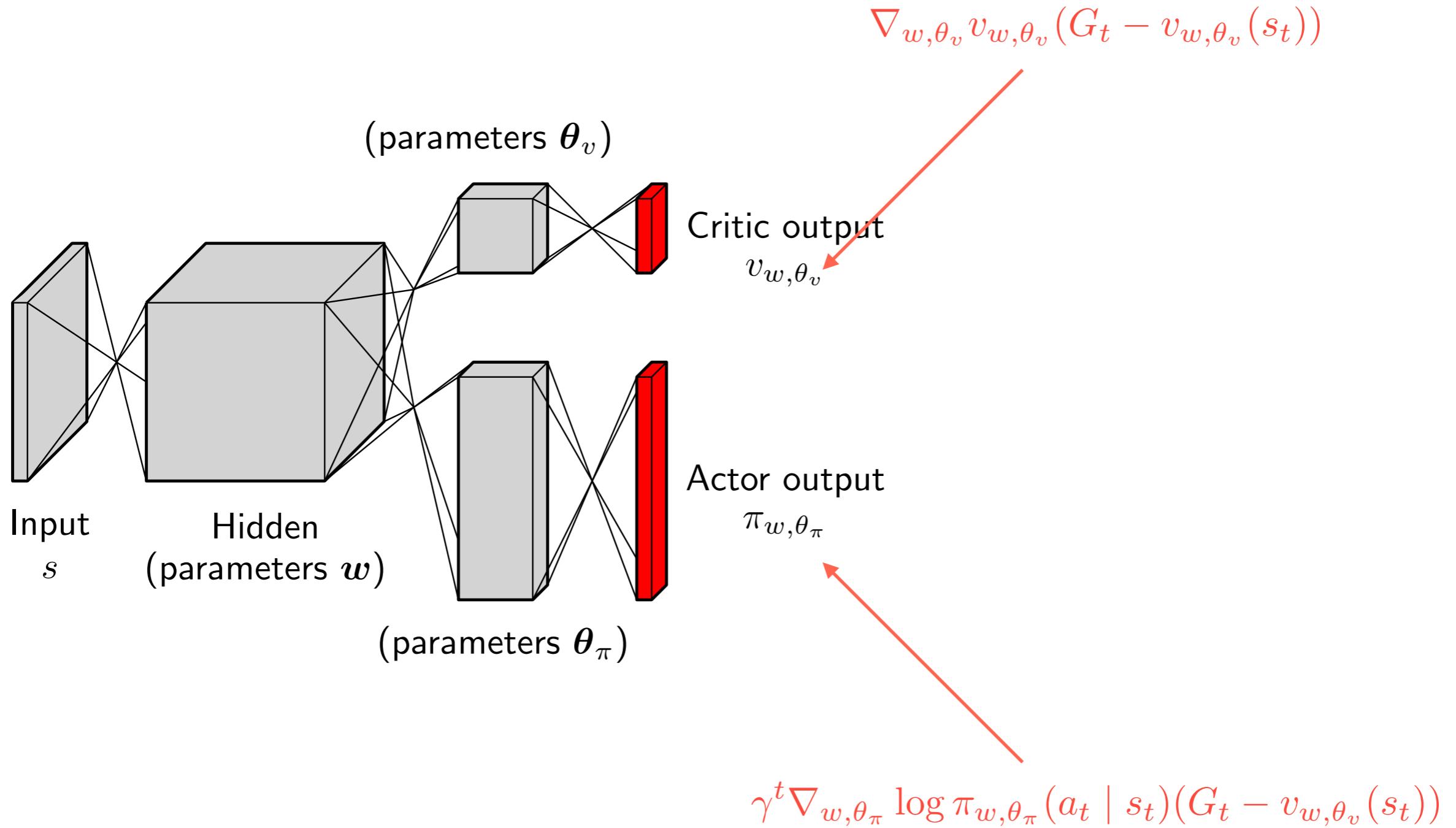
Actor-critic architecture

- The AC architecture comprises two components:
 - An **actor**, responsible for executing the policy π_θ
 - A **critic**, responsible for evaluating the policy (computing adv_π)

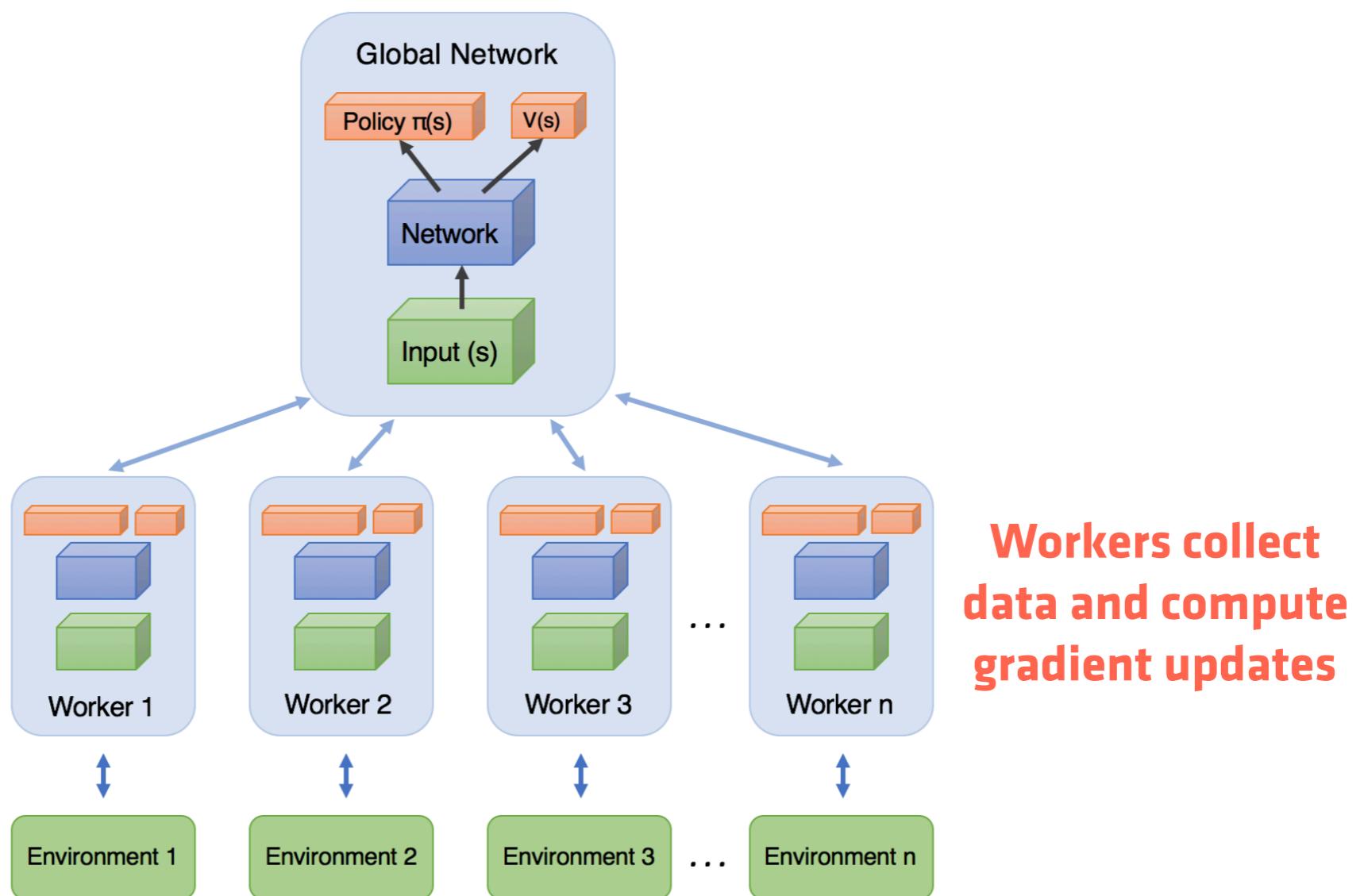
**The two components
are used to estimate
the advantage**



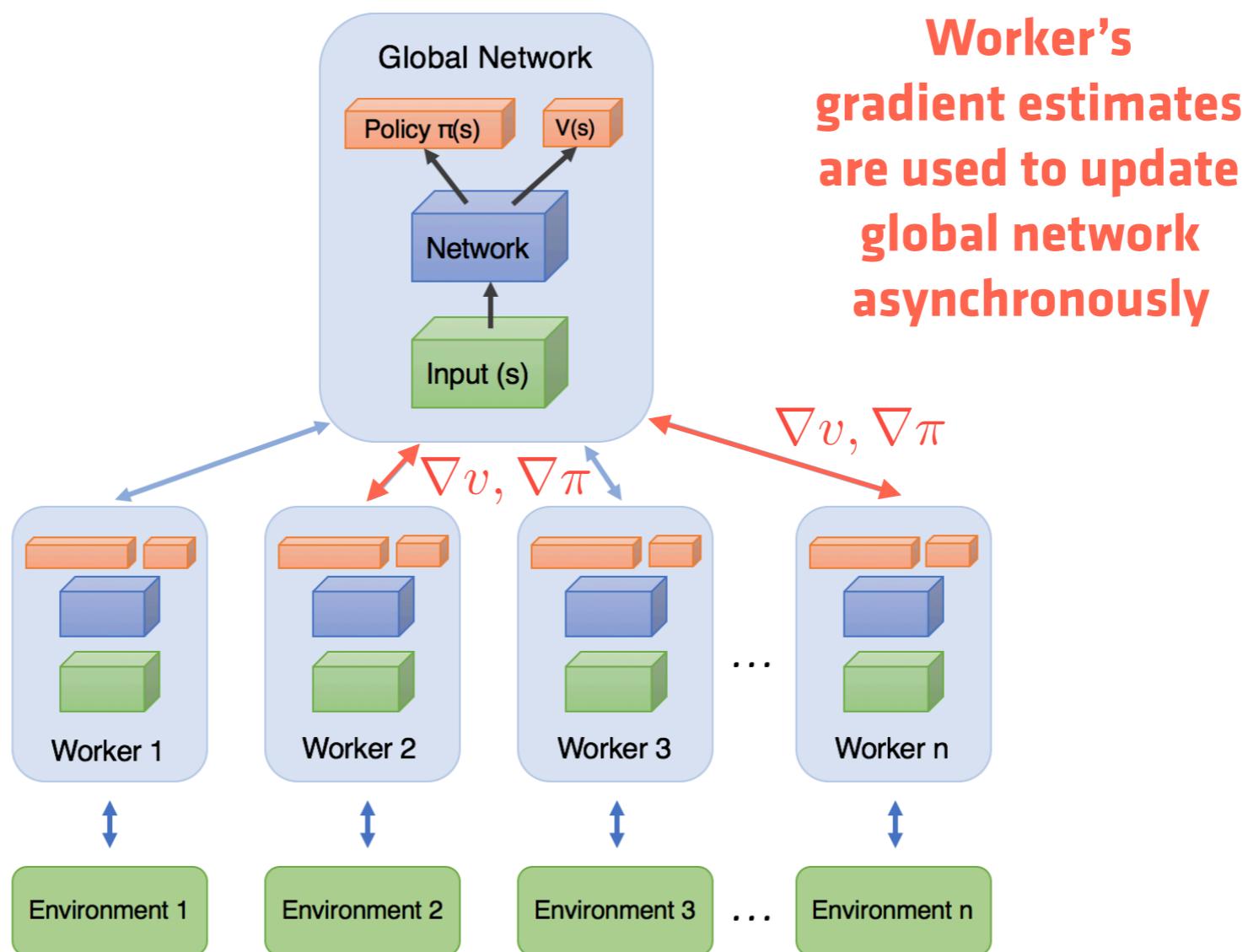
Advantage Actor-Critic



Asynchronous Advantage Actor-Critic (A3C)

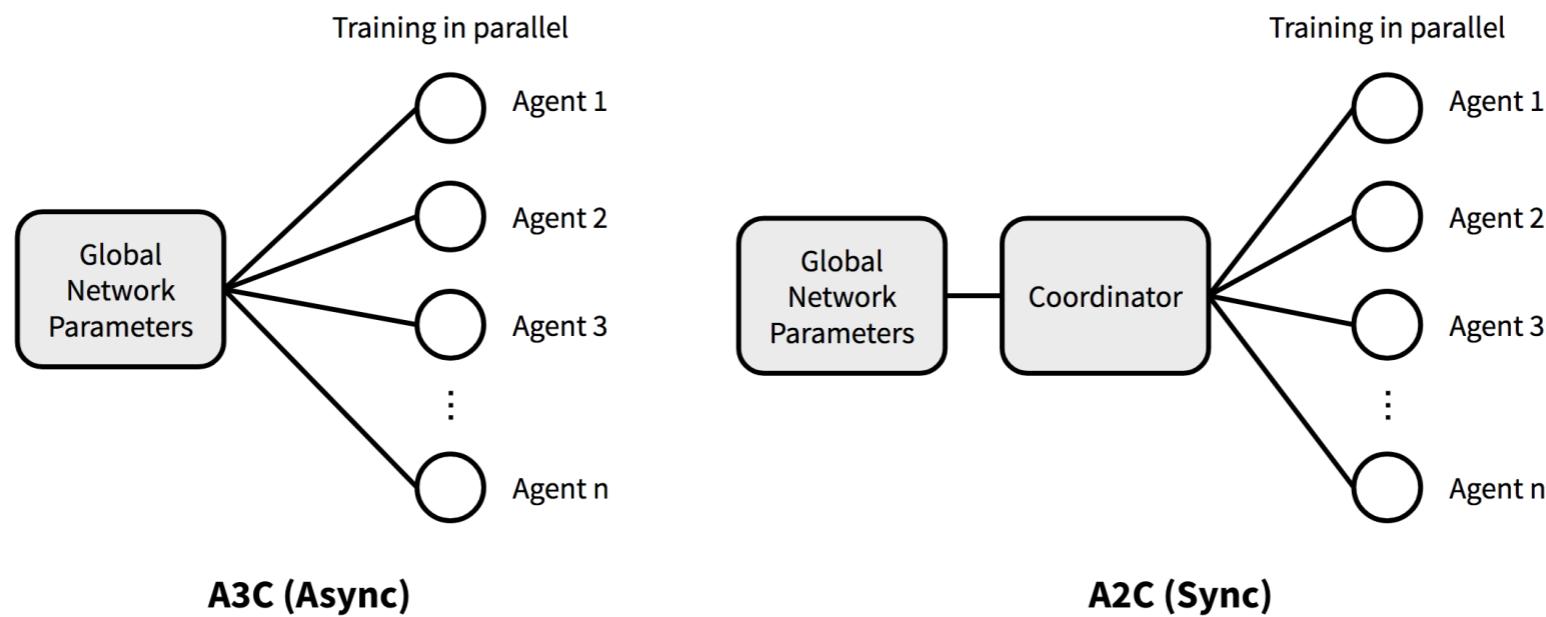


Asynchronous Advantage Actor-Critic (A3C)



Asynchronous Advantage Actor-Critic (A3C)

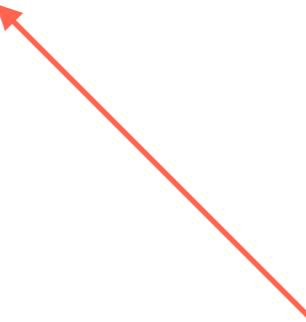
- It is not clear that asynchrony brings an advantage
 - Ongoing work to compare A3C with its synchronous version (A2C)
 - A2C includes a **coordinator module** that ensures that gradient updates are synchronized



Let's take a step back...

How PG methods work

- Start with a parameterized policy
- Gather some data (trajectories) using that policy
- Use the data to estimate the advantage
- Update policy parameters using the gradient
- Repeat



At this point,
what happens to
the data?

How PG methods work

- Old data is “discarded”
 - Old trajectories may be unlikely under the updated policy
 - Old trajectories provide poor estimate to the advantage under updated policy

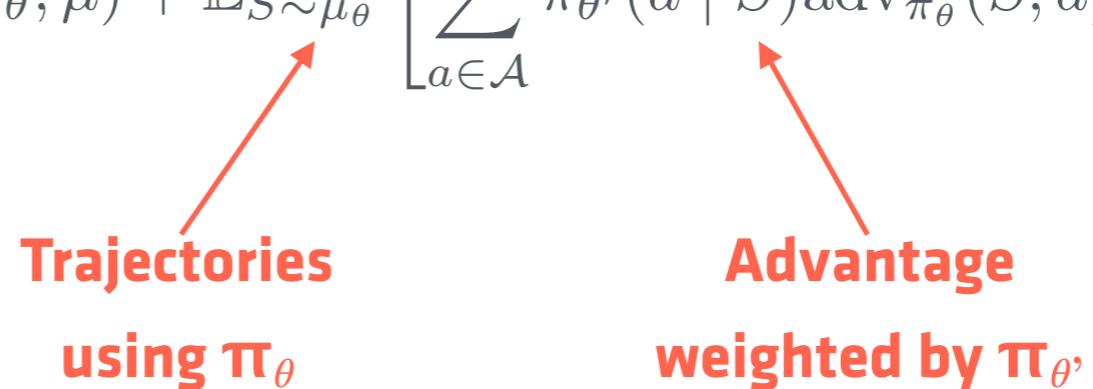


**Not very
data
efficient**

Alternative optimization

- Recall that policy gradient methods arise from the optimization of $J(\pi; \mu)$
- Given two policies, π_θ and $\pi_{\theta'}$, it is possible to show that

$$J(\pi_{\theta'}; \mu) = J(\pi_\theta; \mu) + \mathbb{E}_{S \sim \mu_\theta} \left[\sum_{a \in \mathcal{A}} \pi_{\theta'}(a | S) \text{adv}_{\pi_\theta}(S, a) \right]$$



Trajectories
using π_θ

Advantage
weighted by $\pi_{\theta'}$

Alternative optimization

- Recall that policy gradient methods arise from the optimization of $J(\pi; \mu)$
- Given two policies, π_θ and $\pi_{\theta'}$, it is possible to show that

$$J(\pi_{\theta'}; \mu) = J(\pi_\theta; \mu) + \mathbb{E}_{S \sim \mu_\theta} \left[\sum_{a \in \mathcal{A}} \pi_{\theta'}(a | S) \text{adv}_{\pi_\theta}(S, a) \right]$$

if π_θ and $\pi_{\theta'}$ are “close”

- We can thus optimize $J(\pi_{\theta'}; \mu)$ by maximizing the expectation on the r.h.s.

Trust region policy optimization

- TRPO thus consists of solving the optimization problem

$$\max_{\theta} \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid S) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, a) \right]$$

subject to $\mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}} [\text{KL}(\pi_{\theta_{\text{old}}}(\cdot \mid S), \pi_{\theta}(\cdot \mid S))] < \delta$ **Trust region**

- Can be solved using standard optimization
- How do we compute the expectation in the objective?

Estimating the expectation

- We have that

$$\mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid S) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, a) \right] = \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}(A \mid S)}{\pi_{\theta_{\text{old}}}(A \mid S)} \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) \right]$$

↑
**Same trajectories
used in standard
PG algorithms**
↑
**Importance
sampling
weight**

Estimating the expectation

- We have that

$$\mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid S) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, a) \right] = \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}(A \mid S)}{\pi_{\theta_{\text{old}}}(A \mid S)} \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) \right]$$

- Right hand side can be estimated from the trajectories
- Interesting fact:
 - If you differentiate the r.h.s. with respect to θ , you get

$$\mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\frac{\nabla \pi_{\theta}(A \mid S)}{\pi_{\theta_{\text{old}}}(A \mid S)} \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) \right]_{\theta=\theta_{\text{old}}} = \nabla_{\theta} J(\theta_{\text{old}}; \mu)$$

Relation to PG

- If instead of KL divergence we use an Euclidean constraint, i.e.

$$\begin{aligned} \max_{\theta} & \quad \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a \mid S) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, a) \right] \\ \text{subject to} & \quad \|\theta - \theta_{\text{old}}\|_2^2 < \delta \end{aligned}$$

we recover standard policy gradient

Are we done?

- **DQN**
 - Relatively simple to implement
 - Not very robust
- **Policy gradient**
 - Relatively simple to implement
 - Not very data efficient
 - Sensitive to step-size

Are we done?

- **TRPO**
 - Robust
 - Data efficient
 - Complex to implement
 - Computationally heavy

Proximal policy optimization

- Turn the TRPO optimization problem into an unconstrained optimization problem

$$L(\theta) = \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}(A | S)}{\pi_{\theta_{\text{old}}}(A | S)} \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) - \beta \text{KL}(\pi_{\theta_{\text{old}}}(\cdot | S), \pi_{\theta}(\cdot | S)) \right]$$

- We could run SGD on the loss above
- However, β should be **adjusted as learning progresses**

Proximal policy optimization

- Alternatively, We could modify the loss to discourage big policy changes

$$L(\theta) = \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\min \left(r_t(\theta) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) \right) \right]$$

$$r(\theta) = \frac{\pi_\theta(A | S)}{\pi_{\theta_{\text{old}}}(A | S)}$$

Doesn't allow
ratio to grow
too big

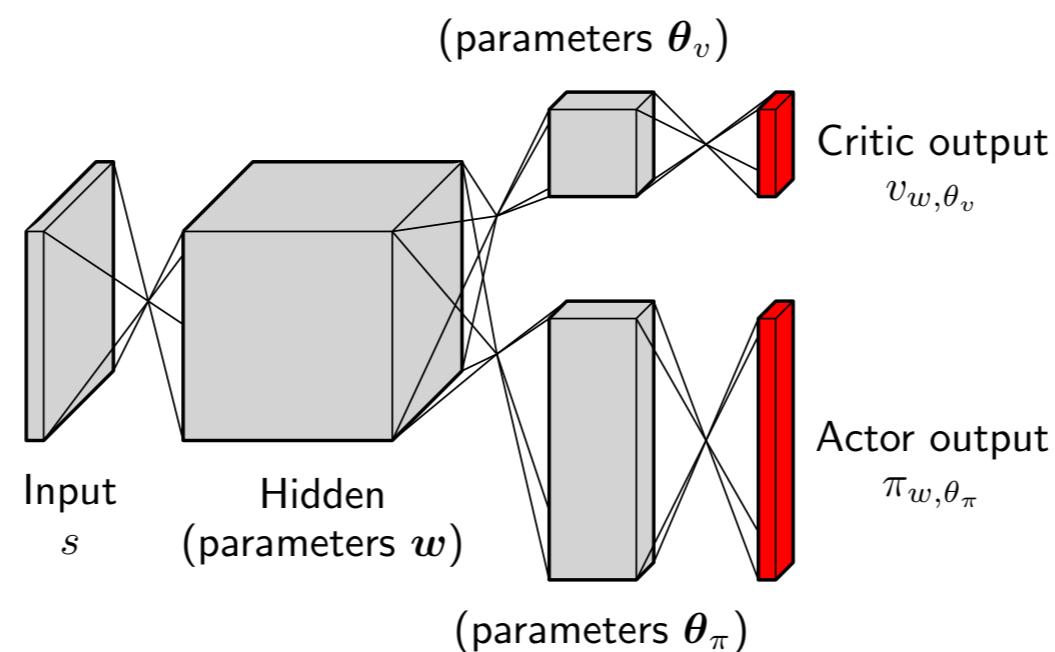
Discourages
large policy
changes

Proximal policy optimization

- Alternatively, We could modify the loss to discourage big policy changes

$$L(\theta) = \mathbb{E}_{S \sim \mu_{\theta_{\text{old}}}, A \sim \pi_{\theta_{\text{old}}}} \left[\min \left(r_t(\theta) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \text{adv}_{\pi_{\theta_{\text{old}}}}(S, A) \right) \right]$$

- Works better in practice than adaptive β
- Similar network architecture than standard PG/AC methods



Outline of the lecture

- **Part I: RL Primer**
 - The RL Problem
 - Markov Decision Process - A Model for RL Problems
 - Optimality & Dynamic Programming
 - Monte Carlo Approaches
 - Temporal Difference Learning
 - The Policy Gradient Theorem

Outline of the lecture

- **Part II: Deep RL**
 - From RL to Deep RL
 - DQN
 - Deep advantage actor-critic methods
 - Trust region methods

Conclusion

- Deep learning is an active area of research
- Many recent developments rely on combining “old” ideas
- Many exploratory works:
 - Algorithmic
 - Architectural
 - Domains



Thank you!

References

- Hasselt, H., Guez, A., and Silver, D. “Deep reinforcement learning with double Q-Learning.” In **Proc. 30th AAAI Conf. Artificial Intelligence**, pp. 2094-2100, 2016. **(DDQN)**
- Mnih, V., Badia, A., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. “Asynchronous methods for deep reinforcement learning.” In **Proc. 33rd Int. Conf. Machine Learning**, pp. 1928-1937, 2016. **(A3C)**
- Mnih, V., et al. “Human-level control through deep reinforcement learning.” **Nature**, 518:529-533, 2015. **(DQN)**
- Schaul, T., Quan, J., Antonoglou, I., and Silver., D. “Prioritized experience replay.” arXiv preprint arXiv:1511.05952, 2015. **(Prioritized experience replay)**
- Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. “Trust Region Policy Optimization.” In **Proc. 32nd Int. Conf. Machine Learning**, pp. 1889-1897, 2015. **(TRPO)**
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. “Proximal Policy Optimization Algorithms.” arXiv preprint arXiv:1707:06347, 2017. **(PPO)**
- Sutton, R., and Barto, A. **Reinforcement Learning: An Introduction**, 2nd Edition, MIT Press, 2018.
- Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. “Dueling network architectures for deep reinforcement learning.” In **Proc. 33rd Int. Conf. Machine Learning**, pp. 1995-2003, 2016. **(Dueling architecture)**