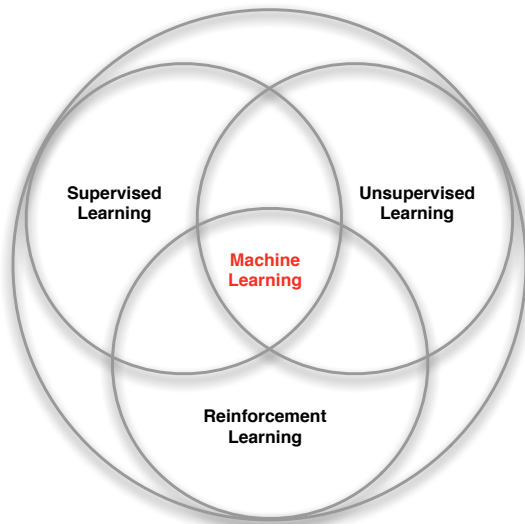


# Branches of Machine Learning



# Characteristics of Reinforcement Learning

What makes reinforcement learning different from other machine learning paradigms?

- There is no supervisor, only a *reward* signal
- Feedback is delayed, not instantaneous
- Time really matters (sequential, non i.i.d data)
- Agent's actions affect the subsequent data it receives

# Examples of Reinforcement Learning

- Fly stunt manoeuvres in a helicopter
- Defeat the world champion at Backgammon
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

# 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**

## Definition (Reward Hypothesis)

*All goals can be described by the maximisation of expected cumulative reward*

Do you agree with this statement?

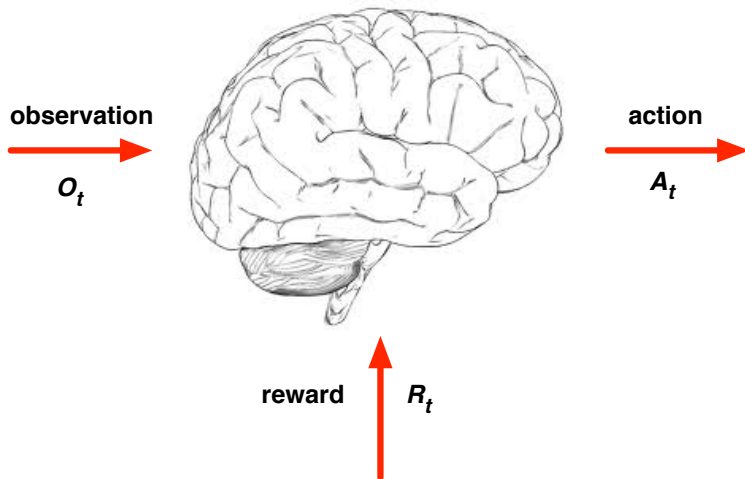
# Examples of Rewards

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

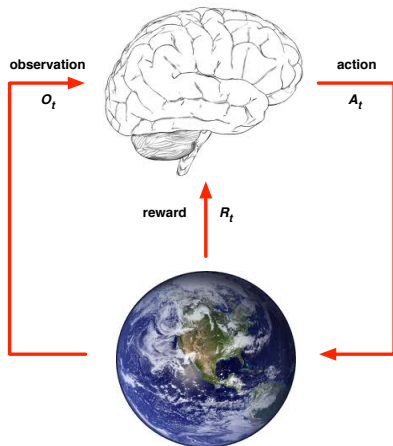
# Sequential Decision Making

- Goal: *select actions to maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Refuelling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

# Agent and Environment



# Agent and Environment



- At each step  $t$  the agent:
  - Executes action  $A_t$
  - Receives observation  $O_t$
  - Receives scalar reward  $R_t$
- The environment:
  - Receives action  $A_t$
  - Emits observation  $O_{t+1}$
  - Emits scalar reward  $R_{t+1}$
- $t$  increments at env. step

# History and State

- The **history** is the sequence of observations, actions, rewards

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

- i.e. all observable variables up to time  $t$
- i.e. the sensorimotor stream of a robot or embodied agent
- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- **State** is the information used to determine what happens next
- Formally, state is a function of the history:

$$S_t = f(H_t)$$

# Information State

An **information state** (a.k.a. **Markov state**) contains all useful information from the history.

## Definition

A state  $S_t$  is **Markov** if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

- “The future is independent of the past given the present”

$$H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$$

- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future
- The environment state  $S_t^e$  is Markov
- The history  $H_t$  is Markov

# Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behaviour function
  - Value function: how good is each state and/or action
  - Model: agent's representation of the environment

# Policy

- A **policy** is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy:  $a = \pi(s)$
- Stochastic policy:  $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

# Value Function

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

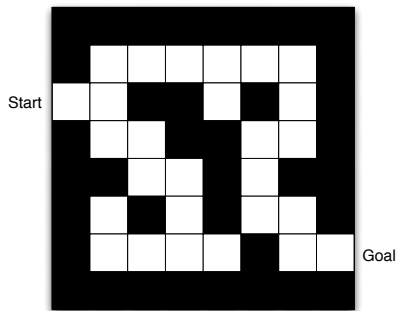
# Model

- A **model** predicts what the environment will do next
- $\mathcal{P}$  predicts the next state
- $\mathcal{R}$  predicts the next (immediate) reward, e.g.

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

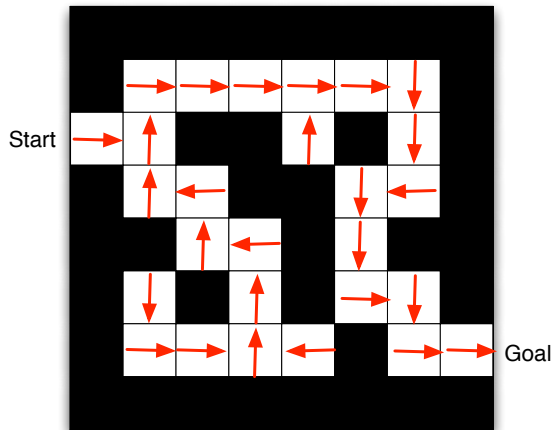
$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

# Maze Example



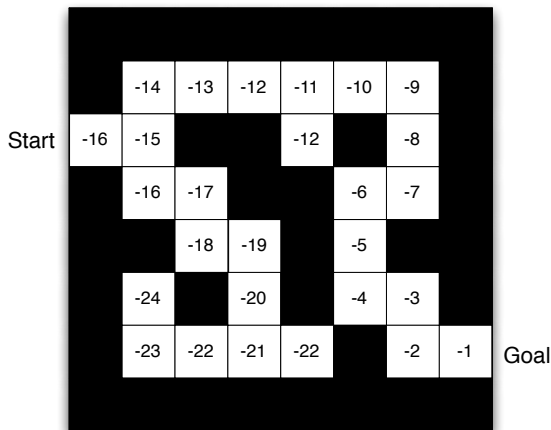
- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

# Maze Example: Policy



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

# Maze Example: Value Function



- Numbers represent value  $v_{\pi}(s)$  of each state  $s$

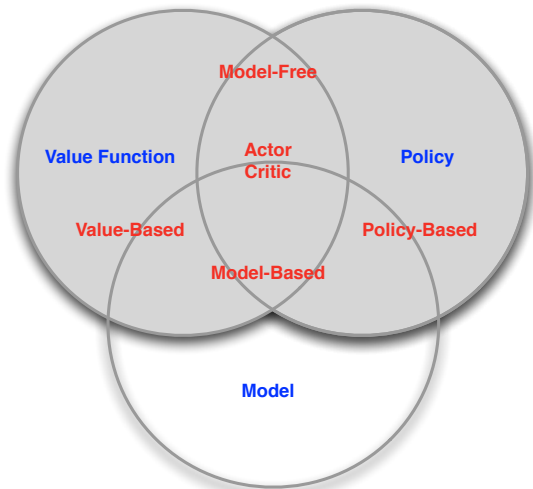
# Categorizing RL agents (1)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function
- Actor Critic
  - Policy
  - Value Function

## Categorizing RL agents (2)

- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

# RL Agent Taxonomy



# Learning and Planning

Two fundamental problems in sequential decision making

- Reinforcement Learning:

- The environment is initially unknown
- The agent interacts with the environment
- The agent improves its policy

- Planning:

- A model of the environment is known
- The agent performs computations with its model (without any external interaction)
- The agent improves its policy
- a.k.a. deliberation, reasoning, introspection, pondering, thought, search

# Exploration and Exploitation (1)

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy
- From its experiences of the environment
- Without losing too much reward along the way

## Exploration and Exploitation (2)

- *Exploration* finds more information about the environment
- *Exploitation* exploits known information to maximise reward
- It is usually important to explore as well as exploit

# Examples

## ■ Restaurant Selection

**Exploitation** Go to your favourite restaurant

**Exploration** Try a new restaurant

## ■ Online Banner Advertisements

**Exploitation** Show the most successful advert

**Exploration** Show a different advert

## ■ Oil Drilling

**Exploitation** Drill at the best known location

**Exploration** Drill at a new location

## ■ Game Playing

**Exploitation** Play the move you believe is best

**Exploration** Play an experimental move

# Prediction and Control

- Prediction: evaluate the future
  - Given a policy
- Control: optimise the future
  - Find the best policy

# Introduction to MDPs

- *Markov decision processes* formally describe an environment for reinforcement learning
- Where the environment is *fully observable*
- i.e. The current *state* completely characterises the process
- Almost all RL problems can be formalised as MDPs, e.g.
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

# Markov Property

“The future is independent of the past given the present”

## Definition

A state  $S_t$  is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

# State Transition Matrix

For a Markov state  $s$  and successor state  $s'$ , the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P} [S_{t+1} = s' \mid S_t = s]$$

State transition matrix  $\mathcal{P}$  defines transition probabilities from all states  $s$  to all successor states  $s'$ ,

$$\mathcal{P} = \begin{matrix} & \text{to} \\ \text{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{matrix}$$

where each row of the matrix sums to 1.

# Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states  $S_1, S_2, \dots$  with the Markov property.

## Definition

A *Markov Process* (or *Markov Chain*) is a tuple  $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$  is a (finite) set of states
- $\mathcal{P}$  is a state transition probability matrix,  
$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$$

# Markov Reward Process

A Markov reward process is a Markov chain with values.

## Definition

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

- $\mathcal{S}$  is a finite set of states
- $\mathcal{P}$  is a state transition probability matrix,  
 $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$
- $\mathcal{R}$  is a reward function,  $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- $\gamma$  is a discount factor,  $\gamma \in [0, 1]$

# Return

## Definition

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

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

- The *discount*  $\gamma \in [0, 1]$  is the present value of future rewards
- The value of receiving reward  $R$  after  $k + 1$  time-steps is  $\gamma^k R$ .
- This values immediate reward above delayed reward.
  - $\gamma$  close to 0 leads to "myopic" evaluation
  - $\gamma$  close to 1 leads to "far-sighted" evaluation

# Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e.  $\gamma = 1$ ), e.g. if all sequences terminate.

# Value Function

The value function  $v(s)$  gives the long-term value of state  $s$

## Definition

The *state value function*  $v(s)$  of an MRP is the expected return starting from state  $s$

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

# Bellman Equation for MRPs

The value function can be decomposed into two parts:

- immediate reward  $R_{t+1}$
- discounted value of successor state  $\gamma v(S_{t+1})$

$$\begin{aligned}v(s) &= \mathbb{E}[G_t \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]\end{aligned}$$

# Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using matrices,

$$\mathbf{v} = \mathcal{R} + \gamma \mathcal{P} \mathbf{v}$$

where  $\mathbf{v}$  is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

# Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

$$(I - \gamma \mathcal{P}) v = \mathcal{R}$$

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

- Computational complexity is  $O(n^3)$  for  $n$  states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

# Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

## Definition

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

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

# Policies (1)

## Definition

A *policy*  $\pi$  is a distribution over actions given states,

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

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent),  
 $A_t \sim \pi(\cdot|S_t), \forall t > 0$

## Policies (2)

- Given an MDP  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  and a policy  $\pi$
- The state sequence  $S_1, S_2, \dots$  is a Markov process  $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence  $S_1, R_2, S_2, \dots$  is a Markov reward process  $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- where

$$\mathcal{P}_{s,s'}^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}_{ss'}^a$$

$$\mathcal{R}_s^\pi = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}_s^a$$

# Value Function

## Definition

The *state-value function*  $v_{\pi}(s)$  of an MDP is the expected return starting from state  $s$ , and then following policy  $\pi$

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

## Definition

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

$$q_{\pi}(s, a) = \mathbb{E}_{\pi} [G_t \mid S_t = s, A_t = a]$$

# Bellman Expectation Equation

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

The action-value function can similarly be decomposed,

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

# Bellman Expectation Equation (Matrix Form)

The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_{\pi} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} v_{\pi}$$

with direct solution

$$v_{\pi} = (I - \gamma \mathcal{P}^{\pi})^{-1} \mathcal{R}^{\pi}$$

# Optimal Value Function

## Definition

The *optimal state-value function*  $v_*(s)$  is the maximum value function over all policies

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

The *optimal action-value function*  $q_*(s, a)$  is the maximum action-value function over all policies

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

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is “solved” when we know the optimal value fn.

# Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_{\pi}(s) \geq v_{\pi'}(s), \forall s$$

## Theorem

*For any Markov Decision Process*

- *There exists an optimal policy  $\pi_*$  that is better than or equal to all other policies,  $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function,  $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function,  $q_{\pi_*}(s, a) = q_*(s, a)$*

# Finding an Optimal Policy

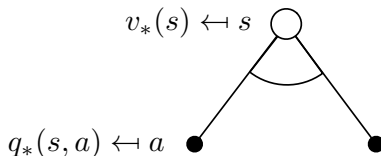
An optimal policy can be found by maximising over  $q_*(s, a)$ ,

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

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

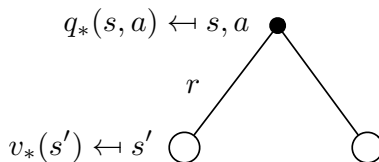
# Bellman Optimality Equation for $v_*$

The optimal value functions are recursively related by the Bellman optimality equations:



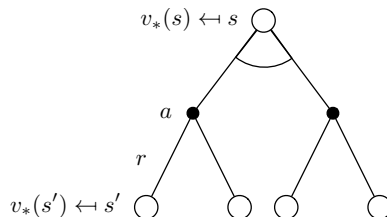
$$v_*(s) = \max_a q_*(s, a)$$

# Bellman Optimality Equation for $Q^*$



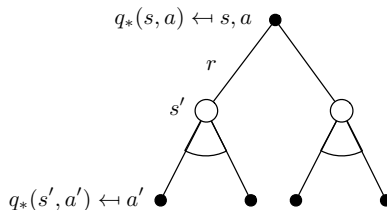
$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

# Bellman Optimality Equation for $V^*$ (2)



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

# Bellman Optimality Equation for $Q^*$ (2)



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

# Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa

# What is Dynamic Programming?

**Dynamic** sequential or temporal component to the problem

**Programming** optimising a “program”, i.e. a policy

- c.f. linear programming
- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

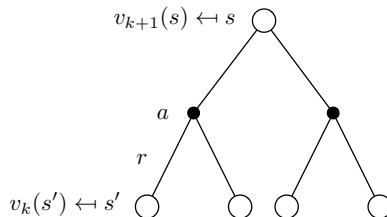
# Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$  and policy  $\pi$
  - or: MRP  $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function  $v_\pi$
- Or for control:
  - Input: MDP  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function  $v_*$
  - and: optimal policy  $\pi_*$

# Iterative Policy Evaluation

- Problem: evaluate a given policy  $\pi$
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration  $k + 1$
  - For all states  $s \in \mathcal{S}$
  - Update  $v_{k+1}(s)$  from  $v_k(s')$
  - where  $s'$  is a successor state of  $s$
- We will discuss *asynchronous* backups later
- Convergence to  $v_\pi$  will be proven at the end of the lecture

# Iterative Policy Evaluation (2)



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

$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

# Evaluating a Random Policy in the Small Gridworld



actions

|    |    |    |    |
|----|----|----|----|
|    | 1  | 2  | 3  |
| 4  | 5  | 6  | 7  |
| 8  | 9  | 10 | 11 |
| 12 | 13 | 14 |    |

$r = -1$   
on all transitions

- Undiscounted episodic MDP ( $\gamma = 1$ )
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is  $-1$  until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

# Iterative Policy Evaluation in Small Gridworld

$v_k$  for the  
Random Policy

$k = 0$

|     |     |     |     |
|-----|-----|-----|-----|
| 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 |

Greedy Policy  
w.r.t.  $v_k$

|   |   |   |   |
|---|---|---|---|
|   | ↔ | ↔ | ↔ |
| ↔ | ↔ | ↔ | ↔ |
| ↔ | ↔ | ↔ | ↔ |
| ↔ | ↔ | ↔ |   |

← 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  |

|   |   |   |   |
|---|---|---|---|
|   | ← | ↔ | ↔ |
| ↑ | ↔ | ↔ | ↔ |
| ↔ | ↔ | ↔ | ↓ |
| ↔ | ↔ | → |   |

$k = 2$

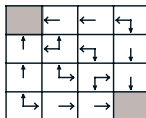
|      |      |      |      |
|------|------|------|------|
| 0.0  | -1.7 | -2.0 | -2.0 |
| -1.7 | -2.0 | -2.0 | -2.0 |
| -2.0 | -2.0 | -2.0 | -1.7 |
| -2.0 | -2.0 | -1.7 | 0.0  |

|   |   |   |   |
|---|---|---|---|
|   | ← | ← | ↔ |
| ↑ | ↖ | ↔ | ↓ |
| ↑ | ↔ | ↘ | ↓ |
| ↔ | → | → |   |

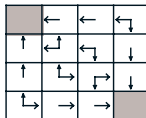
# Iterative Policy Evaluation in Small Gridworld (2)

 $k = 3$ 

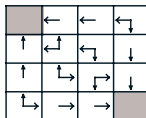
|      |      |      |      |
|------|------|------|------|
| 0.0  | -2.4 | -2.9 | -3.0 |
| -2.4 | -2.9 | -3.0 | -2.9 |
| -2.9 | -3.0 | -2.9 | -2.4 |
| -3.0 | -2.9 | -2.4 | 0.0  |


 $k = 10$ 

|      |      |      |      |
|------|------|------|------|
| 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  |



optimal policy

# How to Improve a Policy

- Given a policy  $\pi$ 
  - **Evaluate** the policy  $\pi$

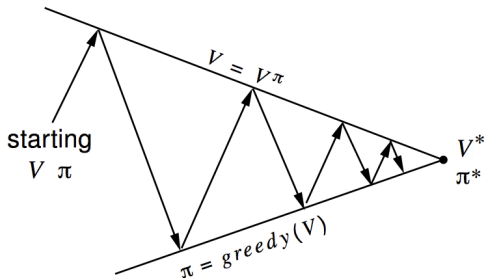
$$v_{\pi}(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

- **Improve** the policy by acting greedily with respect to  $v_{\pi}$

$$\pi' = \text{greedy}(v_{\pi})$$

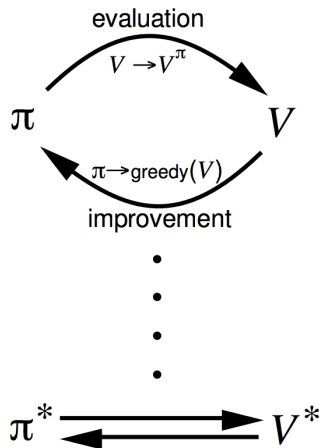
- In Small Gridworld improved policy was optimal,  $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of **policy iteration** always converges to  $\pi^*$

# Policy Iteration



**Policy evaluation** Estimate  $v_\pi$   
Iterative policy evaluation

**Policy improvement** Generate  $\pi' \geq \pi$   
Greedy policy improvement



# Policy Improvement

- Consider a deterministic policy,  $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- This improves the value from any state  $s$  over one step,

$$q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) \geq q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- It therefore improves the value function,  $v_{\pi'}(s) \geq v_{\pi}(s)$

$$\begin{aligned} v_{\pi}(s) &\leq q_{\pi}(s, \pi'(s)) = \mathbb{E}_{\pi'} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma q_{\pi}(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \gamma^2 q_{\pi}(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s] \\ &\leq \mathbb{E}_{\pi'} [R_{t+1} + \gamma R_{t+2} + \dots \mid S_t = s] = v_{\pi'}(s) \end{aligned}$$

## Policy Improvement (2)

- If improvements stop,

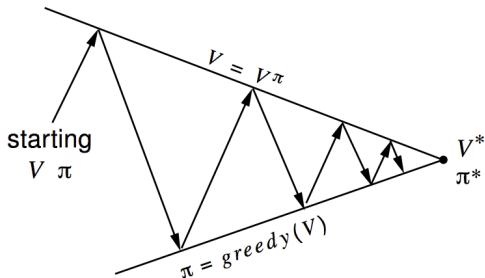
$$q_{\pi}(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_{\pi}(s, a) = q_{\pi}(s, \pi(s)) = v_{\pi}(s)$$

- Then the Bellman optimality equation has been satisfied

$$v_{\pi}(s) = \max_{a \in \mathcal{A}} q_{\pi}(s, a)$$

- Therefore  $v_{\pi}(s) = v_{*}(s)$  for all  $s \in \mathcal{S}$
- so  $\pi$  is an optimal policy

# Generalised Policy Iteration

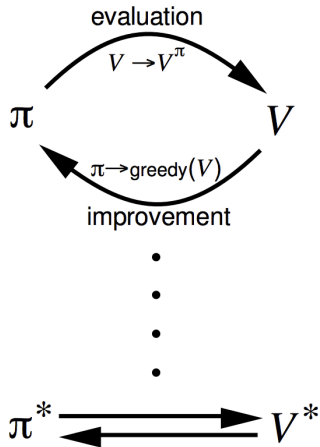


**Policy evaluation** Estimate  $v_\pi$

**Any** policy evaluation algorithm

**Policy improvement** Generate  $\pi' \geq \pi$

**Any** policy improvement algorithm



# Principle of Optimality

Any optimal policy can be subdivided into two components:

- An optimal first action  $A_*$
- Followed by an optimal policy from successor state  $S'$

## Theorem (Principle of Optimality)

*A policy  $\pi(a|s)$  achieves the optimal value from state  $s$ ,  $v_\pi(s) = v_*(s)$ , if and only if*

- *For any state  $s'$  reachable from  $s$*
- *$\pi$  achieves the optimal value from state  $s'$ ,  $v_\pi(s') = v_*(s')$*

# Deterministic Value Iteration

- If we know the solution to subproblems  $v_*(s')$
- Then solution  $v_*(s)$  can be found by one-step lookahead

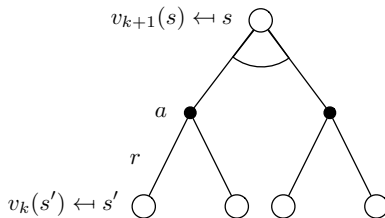
$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

- The idea of value iteration is to apply these updates iteratively
- Intuition: start with final rewards and work backwards
- Still works with loopy, stochastic MDPs

# Value Iteration

- Problem: find optimal policy  $\pi$
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- Using synchronous backups
  - At each iteration  $k + 1$
  - For all states  $s \in \mathcal{S}$
  - Update  $v_{k+1}(s)$  from  $v_k(s')$
- Convergence to  $v_*$  will be proven later
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

# Value Iteration (2)



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

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \mathbf{R}^a + \gamma \mathbf{P}^a \mathbf{v}_k$$

# Synchronous Dynamic Programming Algorithms

| Problem    | Bellman Equation                                         | Algorithm                   |
|------------|----------------------------------------------------------|-----------------------------|
| Prediction | Bellman Expectation Equation                             | Iterative Policy Evaluation |
| Control    | Bellman Expectation Equation + Greedy Policy Improvement | Policy Iteration            |
| Control    | Bellman Optimality Equation                              | Value Iteration             |

- Algorithms are based on state-value function  $v_{\pi}(s)$  or  $v_*(s)$
- Complexity  $O(mn^2)$  per iteration, for  $m$  actions and  $n$  states
- Could also apply to action-value function  $q_{\pi}(s, a)$  or  $q_*(s, a)$
- Complexity  $O(m^2n^2)$  per iteration

## Some Technical Questions

- How do we know that value iteration converges to  $v_*$ ?
- Or that iterative policy evaluation converges to  $v_\pi$ ?
- And therefore that policy iteration converges to  $v_*$ ?
- Is the solution unique?
- How fast do these algorithms converge?
- These questions are resolved by *contraction mapping theorem*

# Value Function Space

- Consider the vector space  $\mathcal{V}$  over value functions
- There are  $|\mathcal{S}|$  dimensions
- Each point in this space fully specifies a value function  $v(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And therefore the backups must converge on a unique solution

## Value Function $\infty$ -Norm

- We will measure distance between state-value functions  $u$  and  $v$  by the  $\infty$ -norm
- i.e. the largest difference between state values,

$$\|u - v\|_{\infty} = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

# Bellman Expectation Backup is a Contraction

- Define the *Bellman expectation backup operator*  $T^\pi$ ,

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$$

- This operator is a  $\gamma$ -contraction, i.e. it makes value functions closer by at least  $\gamma$ ,

$$\begin{aligned} \|T^\pi(u) - T^\pi(v)\|_\infty &= \|(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi u) - (\mathcal{R}^\pi + \gamma \mathcal{P}^\pi v)\|_\infty \\ &= \|\gamma \mathcal{P}^\pi(u - v)\|_\infty \\ &\leq \|\gamma \mathcal{P}^\pi\| \|u - v\|_\infty \\ &\leq \gamma \|u - v\|_\infty \end{aligned}$$

# Contraction Mapping Theorem

## Theorem (Contraction Mapping Theorem)

*For any metric space  $\mathcal{V}$  that is complete (i.e. closed) under an operator  $T(v)$ , where  $T$  is a  $\gamma$ -contraction,*

- *$T$  converges to a unique fixed point*
- *At a linear convergence rate of  $\gamma$*

# Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator  $T^\pi$  has a unique fixed point
- $v_\pi$  is a fixed point of  $T^\pi$  (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on  $v_\pi$
- Policy iteration converges on  $v_*$

# Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator*  $T^*$ ,

$$T^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$$

- This operator is a  $\gamma$ -contraction, i.e. it makes value functions closer by at least  $\gamma$  (similar to previous proof)

$$\|T^*(u) - T^*(v)\|_\infty \leq \gamma \|u - v\|_\infty$$

# Convergence of Value Iteration

- The Bellman optimality operator  $T^*$  has a unique fixed point
- $v_*$  is a fixed point of  $T^*$  (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on  $v_*$

# Monte-Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
  - All episodes must terminate

# Monte-Carlo Policy Evaluation

- Goal: learn  $v_\pi$  from episodes of experience under policy  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

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

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

# First-Visit Monte-Carlo Policy Evaluation

- To evaluate state  $s$
- The **first** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- By law of large numbers,  $V(s) \rightarrow v_\pi(s)$  as  $N(s) \rightarrow \infty$

# Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state  $s$
- **Every** time-step  $t$  that state  $s$  is visited in an episode,
- Increment counter  $N(s) \leftarrow N(s) + 1$
- Increment total return  $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return  $V(s) = S(s)/N(s)$
- Again,  $V(s) \rightarrow v_{\pi}(s)$  as  $N(s) \rightarrow \infty$

# Incremental Mean

The mean  $\mu_1, \mu_2, \dots$  of a sequence  $x_1, x_2, \dots$  can be computed incrementally,

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

# Incremental Monte-Carlo Updates

- Update  $V(s)$  incrementally after episode  $S_1, A_1, R_2, \dots, S_T$
- For each state  $S_t$  with return  $G_t$

$$N(S_t) \leftarrow N(S_t) + 1$$

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

- In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

# Temporal-Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

# MC and TD

- Goal: learn  $v_\pi$  online from experience under policy  $\pi$
- Incremental every-visit Monte-Carlo
  - Update value  $V(S_t)$  toward *actual* return  $G_t$

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value  $V(S_t)$  toward *estimated* return  $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

- $R_{t+1} + \gamma V(S_{t+1})$  is called the *TD target*
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  is called the *TD error*

# Advantages and Disadvantages of MC vs. TD

- TD can learn *before* knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known
- TD can learn *without* the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

# Bias/Variance Trade-Off

- Return  $G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$  is *unbiased* estimate of  $v_\pi(S_t)$
- True TD target  $R_{t+1} + \gamma v_\pi(S_{t+1})$  is *unbiased* estimate of  $v_\pi(S_t)$
- TD target  $R_{t+1} + \gamma V(S_{t+1})$  is *biased* estimate of  $v_\pi(S_t)$
- TD target is much lower variance than the return:
  - Return depends on *many* random actions, transitions, rewards
  - TD target depends on *one* random action, transition, reward

## Advantages and Disadvantages of MC vs. TD (2)

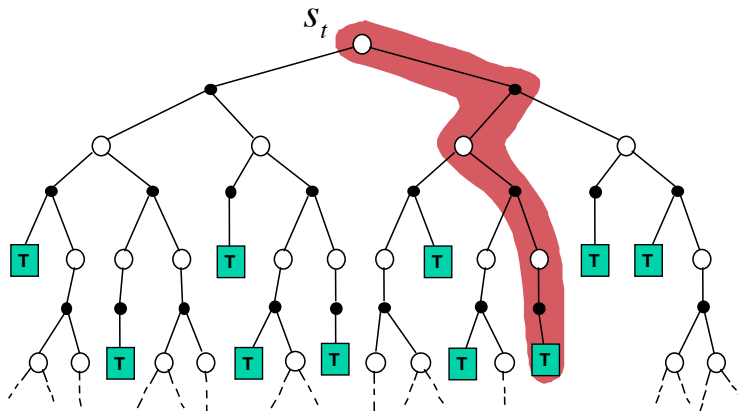
- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to  $v_{\pi}(s)$
  - (but not always with function approximation)
  - More sensitive to initial value

## Advantages and Disadvantages of MC vs. TD (3)

- TD exploits Markov property
  - Usually more efficient in Markov environments
- MC does not exploit Markov property
  - Usually more effective in non-Markov environments

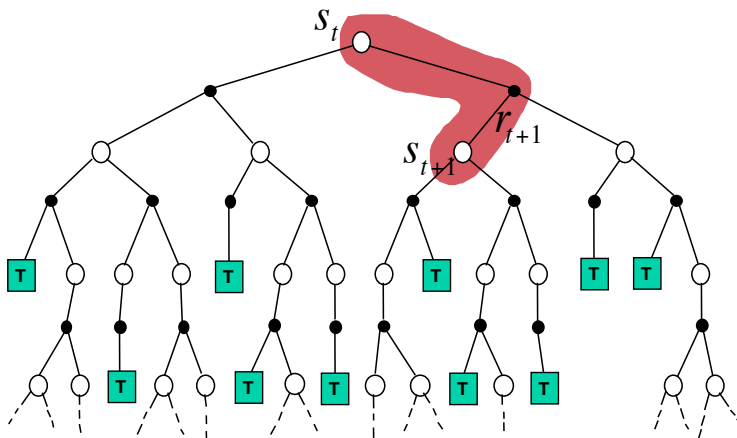
# Monte-Carlo Backup

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



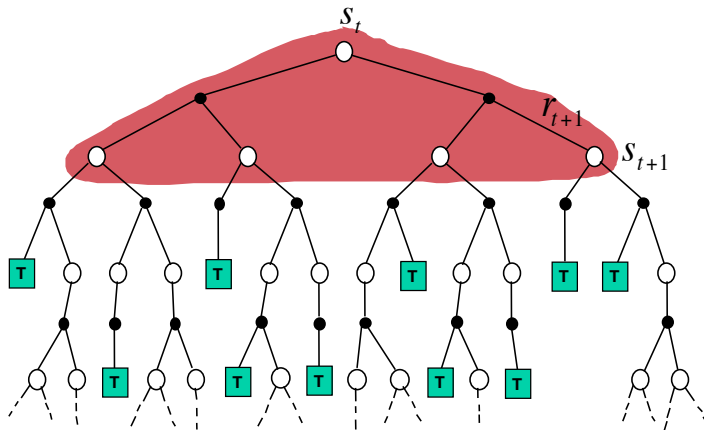
# Temporal-Difference Backup

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



# Dynamic Programming Backup

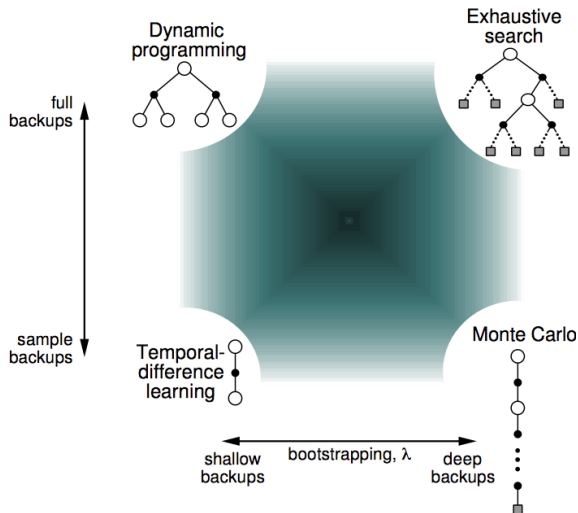
$$V(S_t) \leftarrow \mathbb{E}_{\pi} [R_{t+1} + \gamma V(S_{t+1})]$$



# Bootstrapping and Sampling

- **Bootstrapping**: update involves an estimate
  - MC does not bootstrap
  - DP bootstraps
  - TD bootstraps
- **Sampling**: update samples an expectation
  - MC samples
  - DP does not sample
  - TD samples

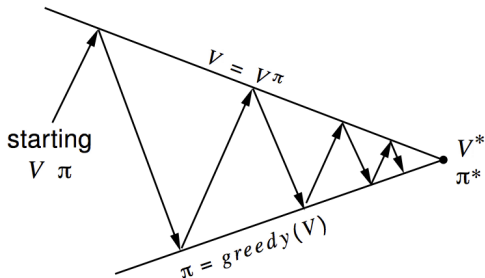
# Unified View of Reinforcement Learning



# On and Off-Policy Learning

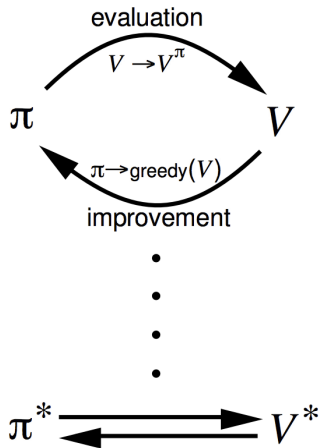
- **On-policy** learning
  - “Learn on the job”
  - Learn about policy  $\pi$  from experience sampled from  $\pi$
- **Off-policy** learning
  - “Look over someone’s shoulder”
  - Learn about policy  $\pi$  from experience sampled from  $\mu$

# Generalised Policy Iteration (Refresher)

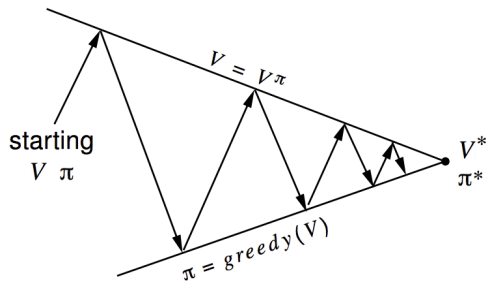


**Policy evaluation** Estimate  $v_\pi$   
 e.g. Iterative policy evaluation

**Policy improvement** Generate  $\pi' \geq \pi$   
 e.g. Greedy policy improvement



# Generalised Policy Iteration With Monte-Carlo Evaluation



**Policy evaluation** Monte-Carlo policy evaluation,  $V = v_\pi$ ?

**Policy improvement** Greedy policy improvement?

# Model-Free Policy Iteration Using Action-Value Function

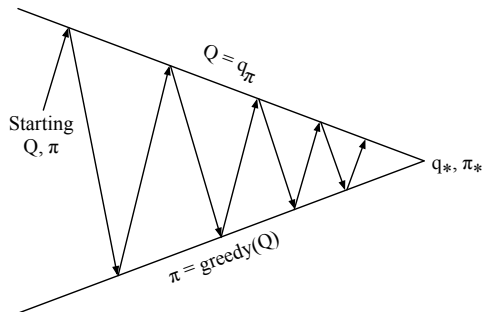
- Greedy policy improvement over  $V(s)$  requires model of MDP

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over  $Q(s, a)$  is model-free

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

# Generalised Policy Iteration with Action-Value Function



**Policy evaluation** Monte-Carlo policy evaluation,  $Q = q_\pi$

**Policy improvement** Greedy policy improvement?

# $\epsilon$ -Greedy Exploration

- Simplest idea for ensuring continual exploration
- All  $m$  actions are tried with non-zero probability
- With probability  $1 - \epsilon$  choose the greedy action
- With probability  $\epsilon$  choose an action at random

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

# $\epsilon$ -Greedy Policy Improvement

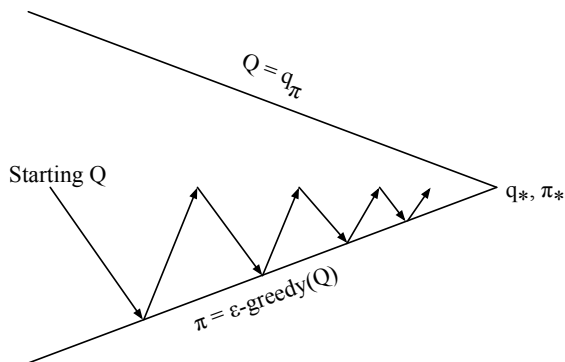
## Theorem

*For any  $\epsilon$ -greedy policy  $\pi$ , the  $\epsilon$ -greedy policy  $\pi'$  with respect to  $q_\pi$  is an improvement,  $v_{\pi'}(s) \geq v_\pi(s)$*

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_{a \in \mathcal{A}} q_\pi(s, a) \\ &\geq \epsilon/m \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s) \end{aligned}$$

Therefore from policy improvement theorem,  $v_{\pi'}(s) \geq v_\pi(s)$

# Monte-Carlo Control



Every episode:

Policy evaluation Monte-Carlo policy evaluation,  $Q \approx q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# GLIE

## Definition

*Greedy in the Limit with Infinite Exploration (GLIE)*

- All state-action pairs are explored infinitely many times,

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = \mathbf{1}(a = \operatorname{argmax}_{a' \in \mathcal{A}} Q_k(s, a'))$$

- For example,  $\epsilon$ -greedy is GLIE if  $\epsilon$  reduces to zero at  $\epsilon_k = \frac{1}{k}$

# GLIE Monte-Carlo Control

- Sample  $k$ th episode using  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- For each state  $S_t$  and action  $A_t$  in the episode,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Improve policy based on new action-value function

$$\epsilon \leftarrow 1/k$$

$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

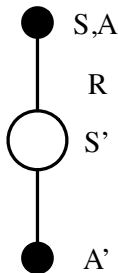
## Theorem

*GLIE Monte-Carlo control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$*

# MC vs. TD Control

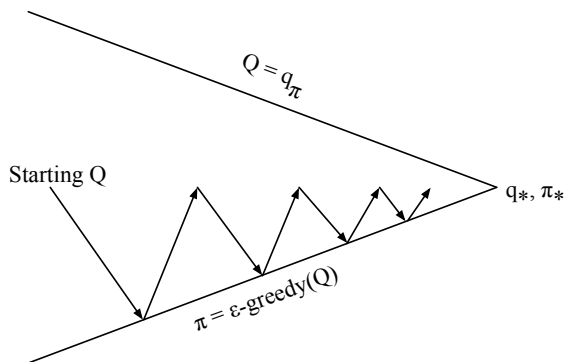
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop
  - Apply TD to  $Q(S, A)$
  - Use  $\epsilon$ -greedy policy improvement
  - Update every time-step

# Updating Action-Value Functions with Sarsa



$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

# On-Policy Control With Sarsa



Every **time-step**:

Policy evaluation **Sarsa**,  $Q \approx q_\pi$

Policy improvement  $\epsilon$ -greedy policy improvement

# Sarsa Algorithm for On-Policy Control

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

    Repeat (for each step of episode):

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$

$S \leftarrow S'; A \leftarrow A';$

    until  $S$  is terminal

# Convergence of Sarsa

## Theorem

*Sarsa converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$ , under the following conditions:*

- *GLIE sequence of policies  $\pi_t(a|s)$*
- *Robbins-Monro sequence of step-sizes  $\alpha_t$*

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

# Off-Policy Learning

- Evaluate target policy  $\pi(a|s)$  to compute  $v_\pi(s)$  or  $q_\pi(s, a)$
- While following behaviour policy  $\mu(a|s)$

$$\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies  $\pi_1, \pi_2, \dots, \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

# Importance Sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X \sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= \mathbb{E}_{X \sim Q} \left[ \frac{P(X)}{Q(X)} f(X) \right]\end{aligned}$$

# Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from  $\mu$  to evaluate  $\pi$
- Weight return  $G_t$  according to similarity between policies
- Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \cdots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

- Update value towards *corrected* return

$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t^{\pi/\mu} - V(S_t) \right)$$

- Cannot use if  $\mu$  is zero when  $\pi$  is non-zero
- Importance sampling can dramatically increase variance

# Importance Sampling for Off-Policy TD

- Use TD targets generated from  $\mu$  to evaluate  $\pi$
- Weight TD target  $R + \gamma V(S')$  by importance sampling
- Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha \left( \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

# Q-Learning

- We now consider off-policy learning of action-values  $Q(s, a)$
- **No** importance sampling is required
- Next action is chosen using behaviour policy  $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action  $A' \sim \pi(\cdot|S_t)$
- And update  $Q(S_t, A_t)$  towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

# Off-Policy Control with Q-Learning

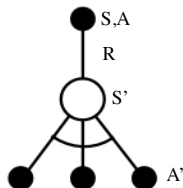
- We now allow both behaviour and target policies to **improve**
- The target policy  $\pi$  is **greedy** w.r.t.  $Q(s, a)$

$$\pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a')$$

- The behaviour policy  $\mu$  is e.g.  **$\epsilon$ -greedy** w.r.t.  $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ &= R_{t+1} + \gamma Q(S_{t+1}, \operatorname{argmax}_{a'} Q(S_{t+1}, a')) \\ &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

# Q-Learning Control Algorithm



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

## Theorem

*Q-learning control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$*

# Q-Learning Algorithm for Off-Policy Control

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)

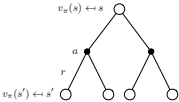
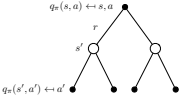
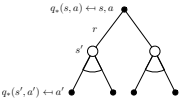
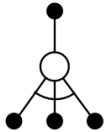
        Take action  $A$ , observe  $R, S'$

$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$

$S \leftarrow S'$ ;

    until  $S$  is terminal

# Relationship Between DP and TD

|                                                  | Full Backup (DP)                                                                                                     | Sample Backup (TD)                                                                                     |
|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Bellman Expectation Equation for $v_{\pi}(s)$    |  <p>Iterative Policy Evaluation</p> |  <p>TD Learning</p> |
| Bellman Expectation Equation for $q_{\pi}(s, a)$ |  <p>Q-Policy Iteration</p>          |  <p>Sarsa</p>       |
| Bellman Optimality Equation for $q_*(s, a)$      |  <p>Q-Value Iteration</p>           |  <p>Q-Learning</p>  |

# Relationship Between DP and TD (2)

| <i>Full Backup (DP)</i>                                                                                                   | <i>Sample Backup (TD)</i>                                                                            |
|---------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| Iterative Policy Evaluation<br>$V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$                                      | TD Learning<br>$V(S) \stackrel{\alpha}{\leftarrow} R + \gamma V(S')$                                 |
| Q-Policy Iteration<br>$Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$                                     | Sarsa<br>$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma Q(S', A')$                                |
| Q-Value Iteration<br>$Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$ | Q-Learning<br>$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$ |

where  $x \stackrel{\alpha}{\leftarrow} y \equiv x \leftarrow x + \alpha(y - x)$

# Policy-Based Reinforcement Learning

- In the last lecture we approximated the value or action-value function using parameters  $\theta$ ,

$$V_{\theta}(s) \approx V^{\pi}(s)$$

$$Q_{\theta}(s, a) \approx Q^{\pi}(s, a)$$

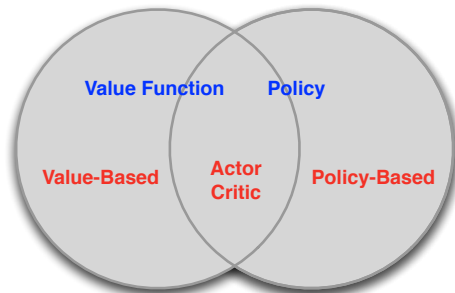
- A policy was generated directly from the value function
  - e.g. using  $\epsilon$ -greedy
- In this lecture we will directly parametrise the **policy**

$$\pi_{\theta}(s, a) = \mathbb{P}[a \mid s, \theta]$$

- We will focus again on **model-free** reinforcement learning

# Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g.  $\epsilon$ -greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



# Advantages of Policy-Based RL

## 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

# Policy Objective Functions

- Goal: given policy  $\pi_\theta(s, a)$  with parameters  $\theta$ , find best  $\theta$
- But how do we measure the quality of a policy  $\pi_\theta$ ?
- In episodic environments we can use the **start value**

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta} [v_1]$$

- In continuing environments we can use the **average value**

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the **average reward per time-step**

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where  $d^{\pi_\theta}(s)$  is **stationary distribution** of Markov chain for  $\pi_\theta$

# Policy Optimisation

- Policy based reinforcement learning is an **optimisation** problem
- Find  $\theta$  that maximises  $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure

# Policy Gradient

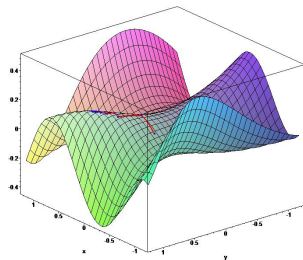
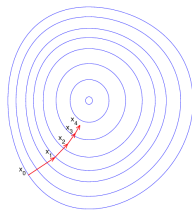
- Let  $J(\theta)$  be any policy objective function
- Policy gradient algorithms search for a *local* maximum in  $J(\theta)$  by ascending the gradient of the policy, w.r.t. parameters  $\theta$

$$\Delta\theta = \alpha \nabla_{\theta} J(\theta)$$

- Where  $\nabla_{\theta} J(\theta)$  is the **policy gradient**

$$\nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and  $\alpha$  is a step-size parameter



# Computing Gradients By Finite Differences

- To evaluate policy gradient of  $\pi_{\theta}(s, a)$
- For each dimension  $k \in [1, n]$ 
  - Estimate  $k$ th partial derivative of objective function w.r.t.  $\theta$
  - By perturbing  $\theta$  by small amount  $\epsilon$  in  $k$ th dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where  $u_k$  is unit vector with 1 in  $k$ th component, 0 elsewhere

- Uses  $n$  evaluations to compute policy gradient in  $n$  dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

# Score Function

- We now compute the policy gradient *analytically*
- Assume policy  $\pi_\theta$  is differentiable whenever it is non-zero
- and we know the gradient  $\nabla_\theta \pi_\theta(s, a)$
- **Likelihood ratios** exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- The **score function** is  $\nabla_\theta \log \pi_\theta(s, a)$

# Softmax Policy

- We will use a softmax policy as a running example
- Weight actions using linear combination of features  $\phi(s, a)^\top \theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s, a) \propto e^{\phi(s, a)^\top \theta}$$

- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - \mathbb{E}_{\pi_\theta} [\phi(s, \cdot)]$$

# Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features  $\mu(s) = \phi(s)^\top \theta$
- Variance may be fixed  $\sigma^2$ , or can also be parametrised
- Policy is Gaussian,  $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_{\theta} \log \pi_{\theta}(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

# One-Step MDPs

- Consider a simple class of **one-step** MDPs
  - Starting in state  $s \sim d(s)$
  - Terminating after one time-step with reward  $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta} [r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \mathcal{R}_{s,a} \\ \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) \mathcal{R}_{s,a} \\ &= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) r] \end{aligned}$$

# Policy Gradient Theorem

- The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward  $r$  with long-term value  $Q^\pi(s, a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

## Theorem

*For any differentiable policy  $\pi_\theta(s, a)$ ,  
for any of the policy objective functions  $J = J_1, J_{avR}$ , or  $\frac{1}{1-\gamma} J_{avV}$ ,  
the policy gradient is*

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$$

# Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return  $v_t$  as an unbiased sample of  $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

## function REINFORCE

Initialise  $\theta$  arbitrarily

**for** each episode  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$  **do**

**for**  $t = 1$  to  $T - 1$  **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

**end for**

**end for**

**return**  $\theta$

**end function**

## Reducing Variance Using a Critic

- Monte-Carlo policy gradient still has high variance
- We use a **critic** to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic** Updates action-value function parameters  $w$
  - Actor** Updates policy parameters  $\theta$ , in direction suggested by critic
- Actor-critic algorithms follow an *approximate* policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

# Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- How good is policy  $\pi_\theta$  for current parameters  $\theta$ ?
- This problem was explored in previous two lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - $TD(\lambda)$
- Could also use e.g. least-squares policy evaluation

# Action-Value Actor-Critic

- Simple actor-critic algorithm based on action-value critic
- Using linear value fn approx.  $Q_w(s, a) = \phi(s, a)^\top w$ 
  - Critic** Updates  $w$  by linear TD(0)
  - Actor** Updates  $\theta$  by policy gradient

**function** QAC

    Initialise  $s, \theta$

    Sample  $a \sim \pi_\theta$

**for** each step **do**

        Sample reward  $r = \mathcal{R}_s^a$ ; sample transition  $s' \sim \mathcal{P}_{s,\cdot}^a$ .

        Sample action  $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

**end for**

**end function**

# Summary of Policy Gradient Algorithms

- The **policy gradient** has many equivalent forms

|                                                                                                                |                              |
|----------------------------------------------------------------------------------------------------------------|------------------------------|
| $\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{v}_t]$ | REINFORCE                    |
| $= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{Q}^w(s, a)]$                     | Q Actor-Critic               |
| $= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{A}^w(s, a)]$                     | Advantage Actor-Critic       |
| $= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta]$                                 | TD Actor-Critic              |
| $= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \delta \mathbf{e}]$                      | TD( $\lambda$ ) Actor-Critic |
| $G_{\theta}^{-1} \nabla_{\theta} J(\theta) = \mathbf{w}$                                                       | Natural Actor-Critic         |

- Each leads a stochastic gradient ascent algorithm
- Critic uses **policy evaluation** (e.g. MC or TD learning) to estimate  $Q^{\pi}(s, a)$ ,  $A^{\pi}(s, a)$  or  $V^{\pi}(s)$