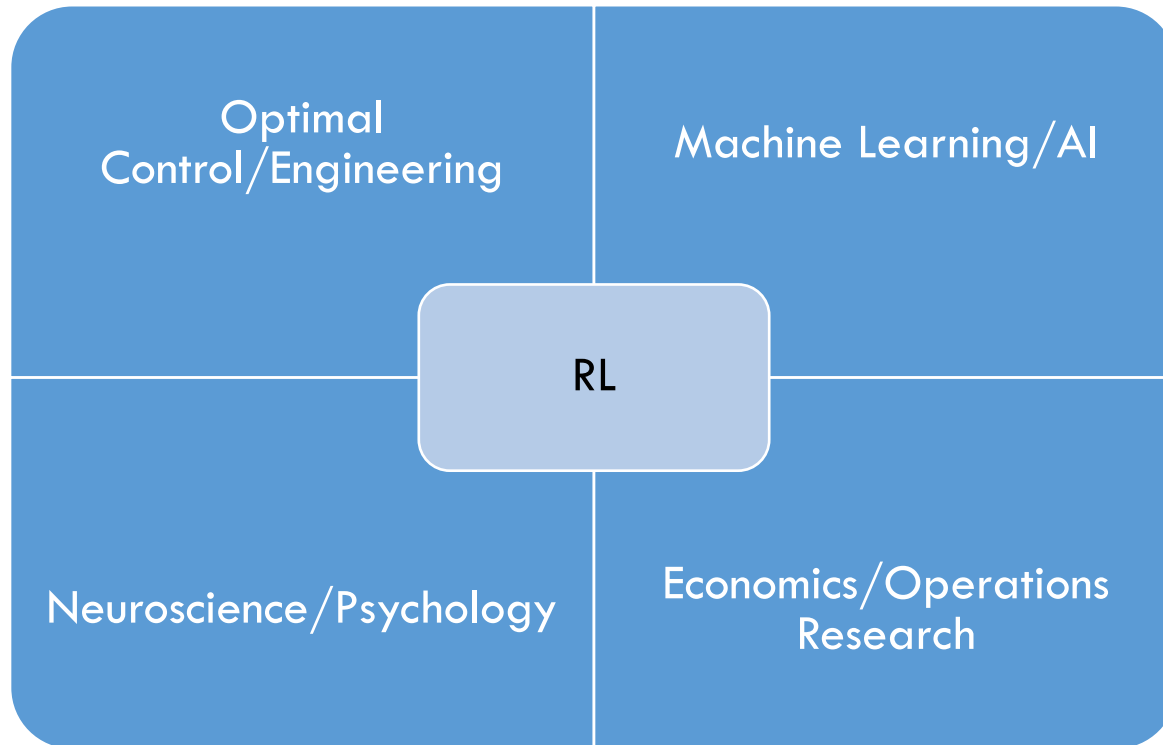

Advanced Prediction Models

Today's Outline

- Complex Decisions
- Reinforcement Learning Basics
 - Markov Decision Process
 - (State Action) Value Function
- Q Learning Algorithm

Complex Decisions

Complex Decisions Making is Everywhere



Control

- Fly drones
- Autonomous driving

Operations

- Retain customers, UX
- Inventory management

Logistics

- Schedule transportation
- Resource allocation

Games

- Chess, Go, Atari

Complex Decisions Making is Everywhere

Computer Go



Brain computer interface



Medical trials



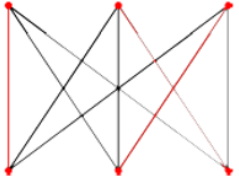
Packets routing



Ads placement



Dynamic allocation



Credit: Sebastien Bubeck

Control

- Fly drones
- Autonomous driving

Operations

- Retain customers, UX
- Inventory management

Logistics

- Schedule transportation
- Resource allocation

Games

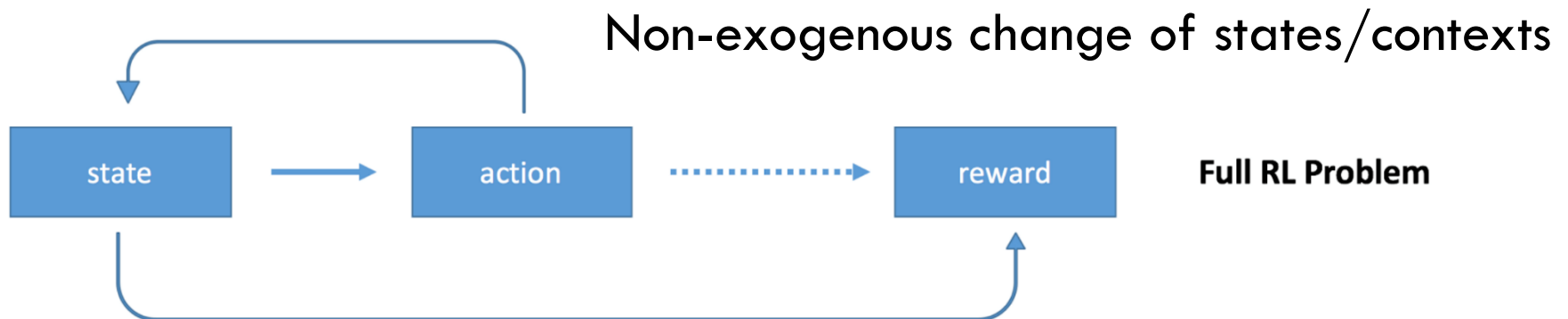
- Chess, Go, Atari

AlphaGo Conquers Go (2016)



¹Reference: DeepMind, March 2016

Need for Reinforcement Learning



RL Overview

- Reinforcement Learning (RL) addresses a version of the problem of **sequential decision making**
- Ingredients:
 - There is an **environment**
 - Within which, an **agent** takes actions
 - This action **influences the future**
 - Agent gets a (potentially **delayed**) feedback signal
- How to select actions to maximize total reward?
- RL provides several sound answers to this question

The Environment

- Sees Agent's action A_t and generates an observation S_{t+1} and a reward R_{t+1}
- Subscript t indexes time. Current observation S_t is called **state**
- Assume the future (at times $t + 1, t + 2, \dots$) is independent of the past ($\dots, t - 2, t - 1$) given the present (t): this is called the Markov assumption

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, S_2, \dots, S_t)$$

- Assume everything relevant is observed

The Agent

- Agent observes R_{t+1}, S_{t+1} and these are not i.i.d. across time
- Agent's objective is to maximize expected total future reward $E[R_{t+1} + \gamma R_{t+2} + \dots]$
- Agent's actions affect what it sees in the future (S_{t+1})
- Maybe better to trade off current reward R_{t+1} to gain more rewards in the future

The Reward

- 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)

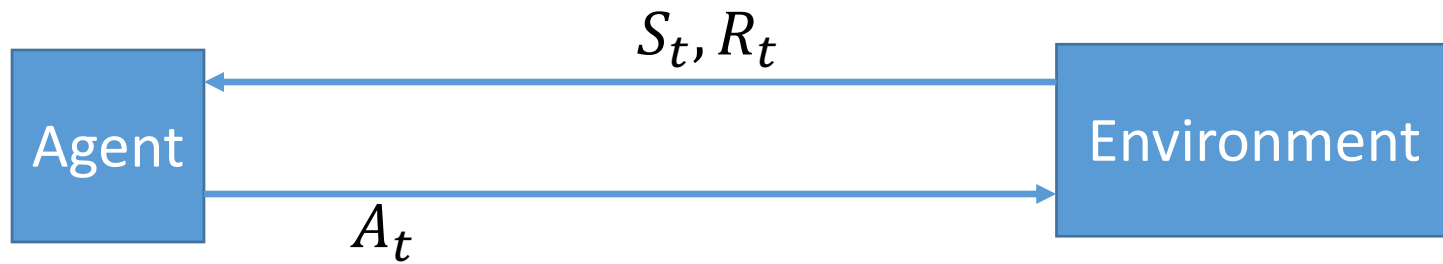
All goals can be described by the maximisation of expected cumulative reward

The Goal

- 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)

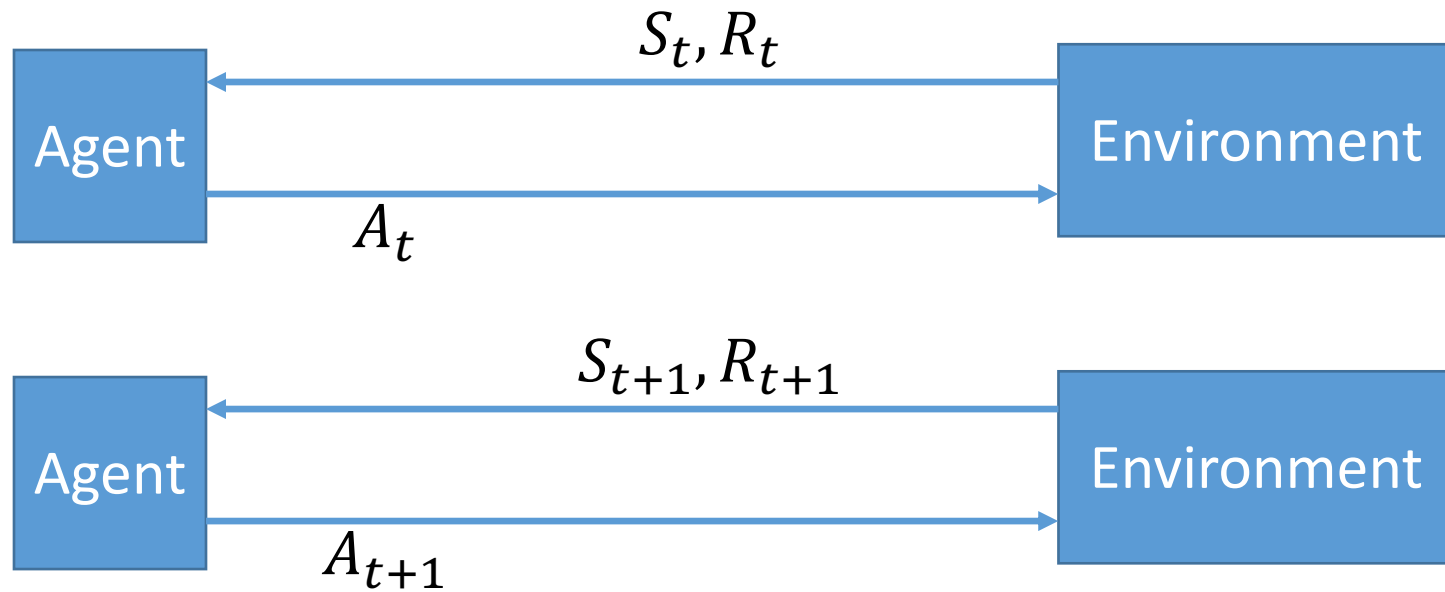
The Interactions

- Pictorially



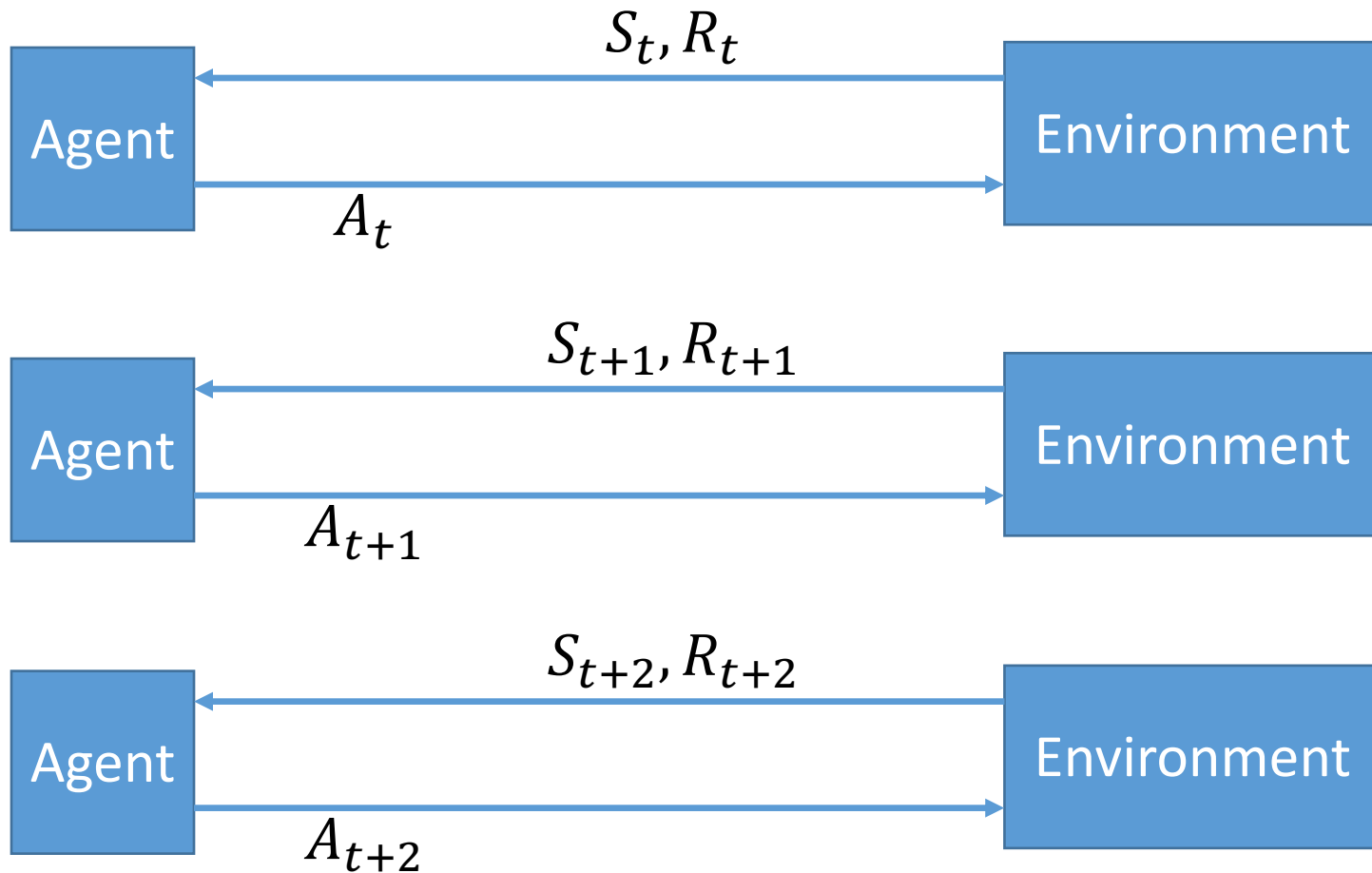
The Interactions

- Pictorially



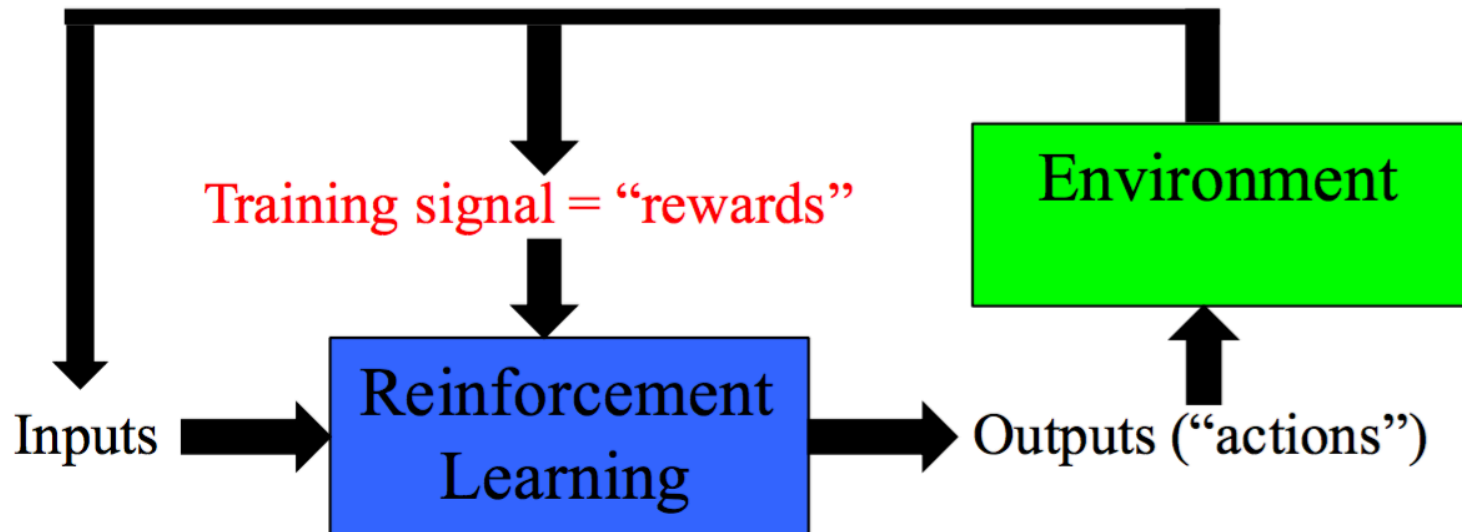
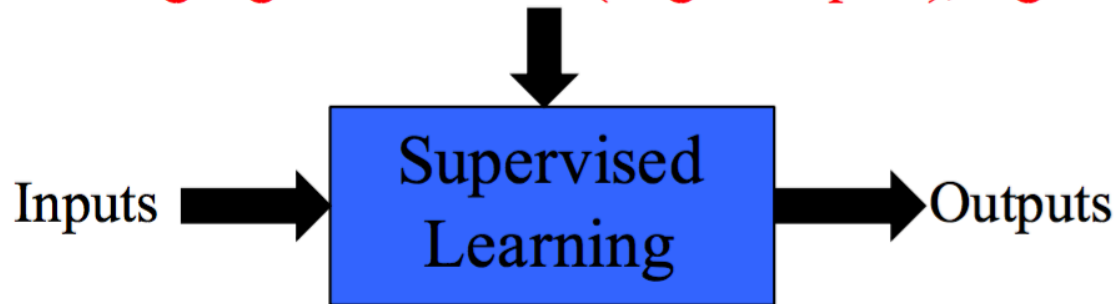
The Interactions

- Pictorially



RL versus other Machine Learning Settings

Training signal = desired (target outputs), e.g. class



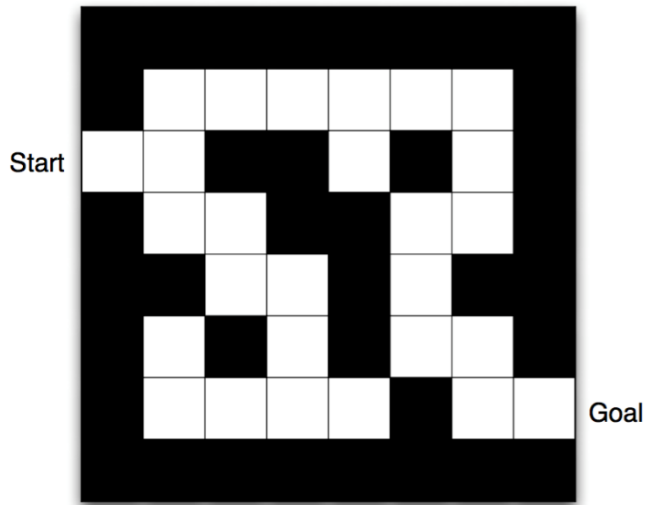
Components of an RL Agent

- 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

Components of RL: Policy

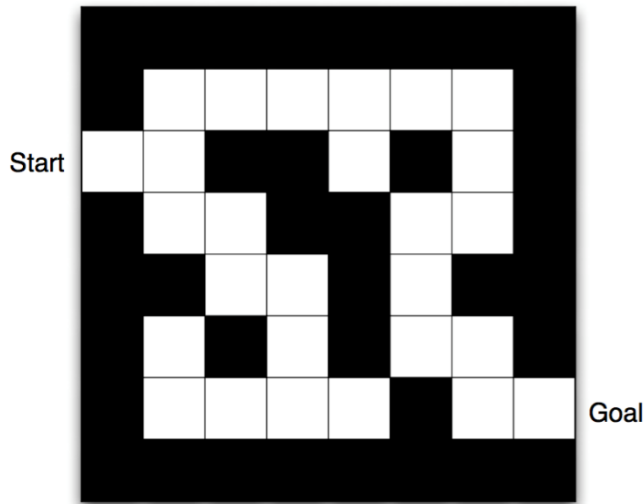
- A **policy** is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy: $a = \pi(s)$
- Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

Components of RL: Policy

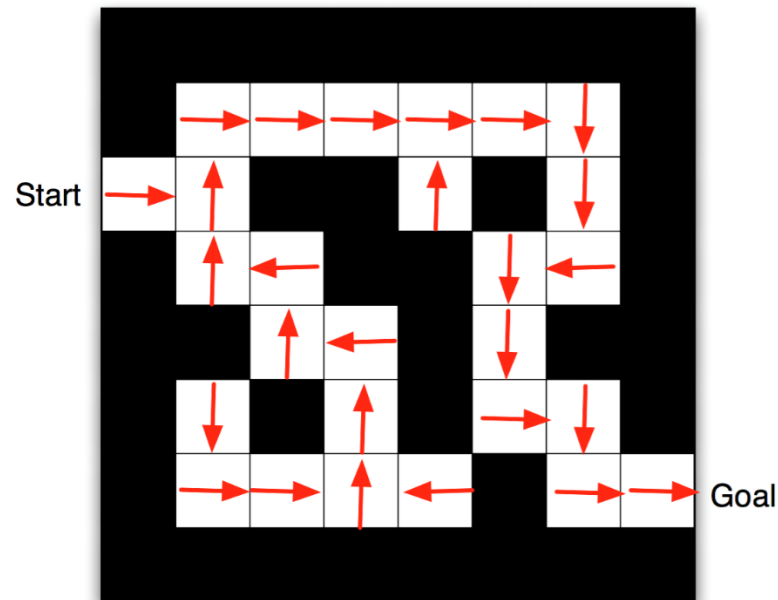


- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

Components of RL: Policy



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

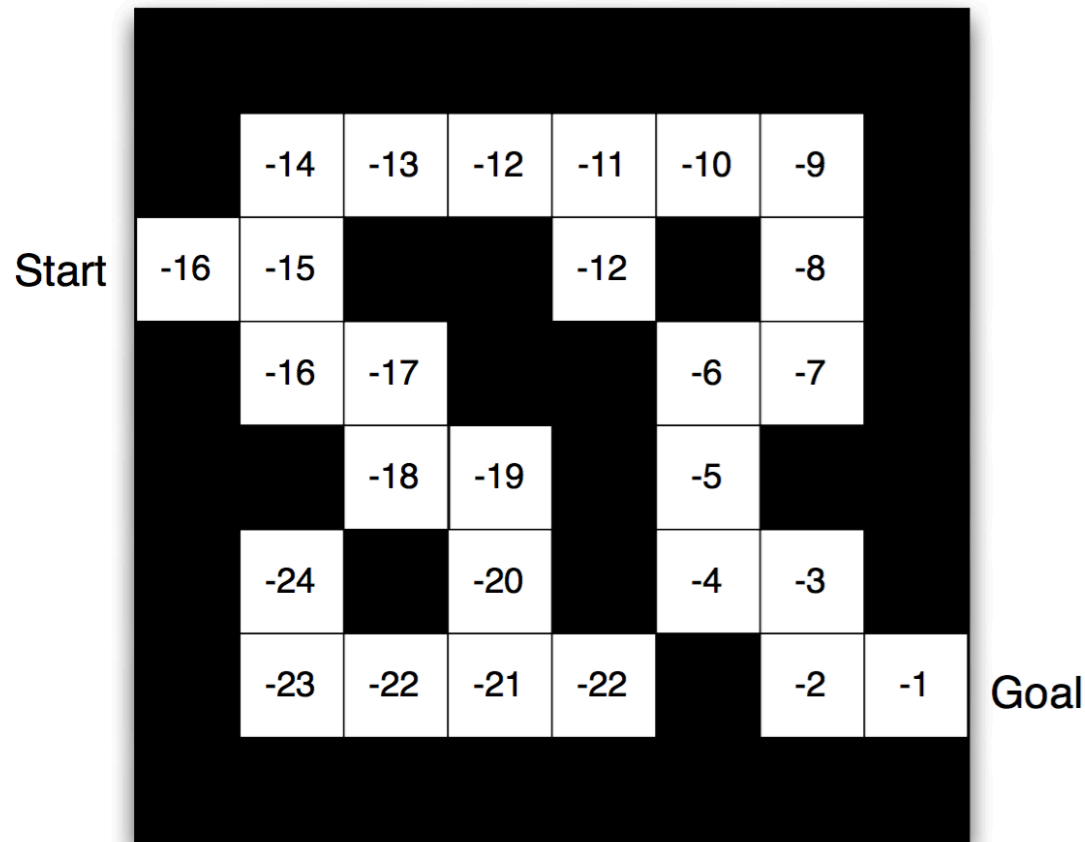


Components of RL: Value Function

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

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

Components of RL: Value Function



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

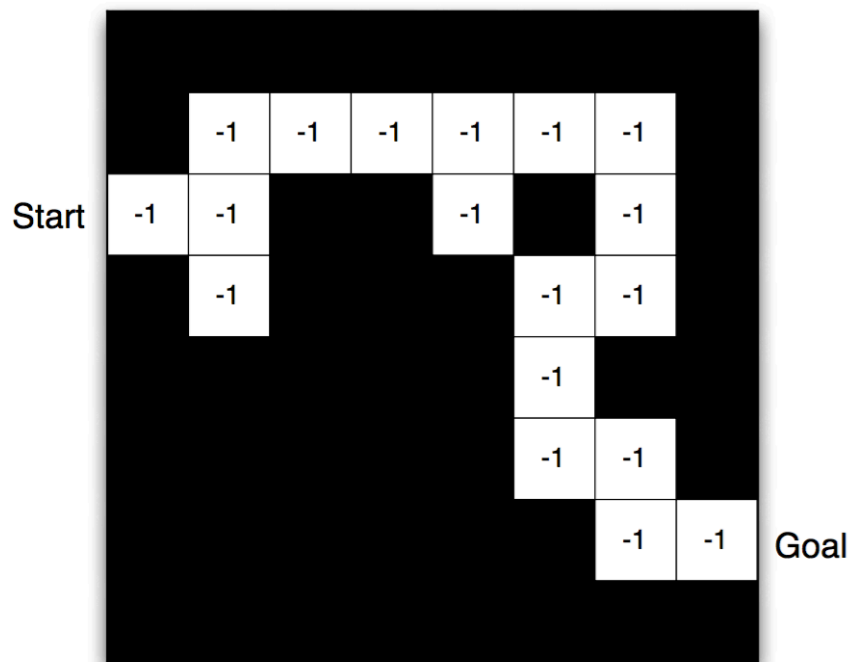
Components of RL: Model

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

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

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

Components of RL: Model



- Dynamics: how actions change the state
- Rewards: how much reward from each state

- Grid layout represents transition model $\mathcal{P}_{ss'}^a$
- Numbers represent immediate reward \mathcal{R}_s^a from each state s (same for all a)

Questions?

Today's Outline

- Complex Decisions
- Reinforcement Learning Basics
 - Markov Decision Process
 - (State Action) Value Function
- Q Learning Algorithm

Components of RL: MDP Framework

- We will now revisit these components **formally**
 - Policy $\pi(a|s)$
 - Value function $v_{\pi}(s)$
 - Model $\mathcal{P}_{ss'}^a$, and \mathcal{R}_s^a
- In the framework of **Markov Decision Processes**
- And then we will address the question of **optimizing** for the best π in **realistic environments**

Towards a Markov Decision Process

- MDPs are a useful way to describe the RL problem
- MDPs can be understood via the following progression
 - Start with a Markov Chain
 - State transitions happen autonomously
 - Add Rewards
 - Becomes a Markov Reward Process
 - Add Actions that influences state transitions
 - Becomes a Markov Decision Process

Markov Chain/Process

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

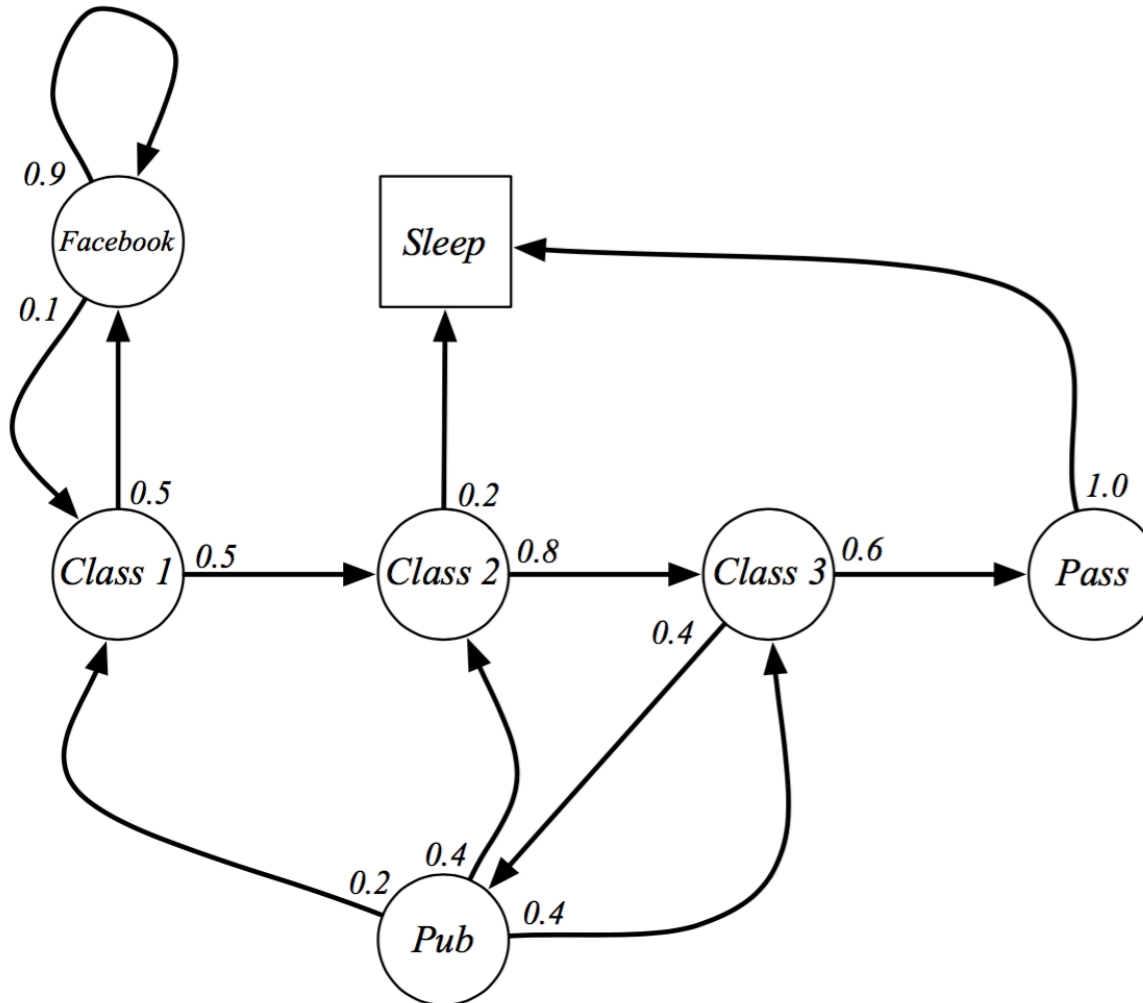
$$\mathcal{P}_{ss'} = \mathbb{P} [S_{t+1} = s' \mid S_t = s]$$

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

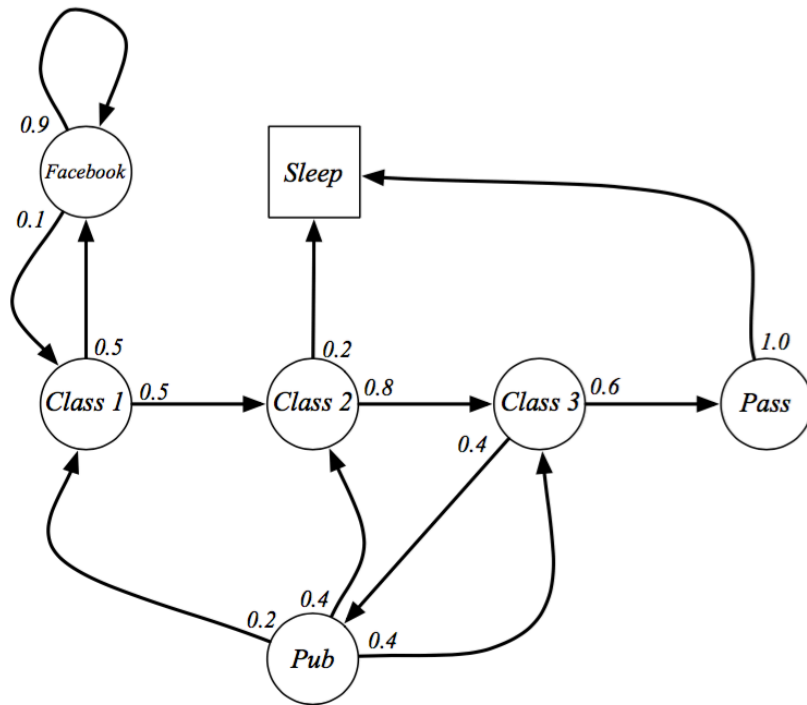
$$\mathcal{P} = \begin{array}{c} \text{to} \\ \left[\begin{array}{ccc} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{array} \right] \\ \text{from} \end{array}$$

where each row of the matrix sums to 1.

Example Markov Chain



Example Markov Chain

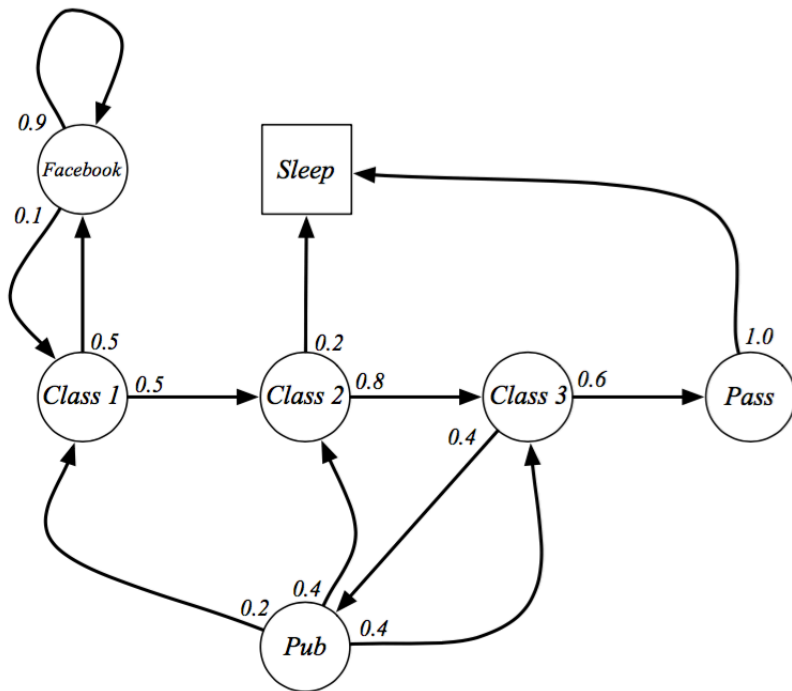


Sample **episodes** for Student Markov Chain starting from $S_1 = C1$

S_1, S_2, \dots, S_T

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Example Markov Chain



$$\mathcal{P} = \begin{matrix} & \begin{matrix} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{matrix} \\ \begin{matrix} C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{matrix} & \begin{bmatrix} & & & & & 0.5 & \\ & 0.5 & & & & & \\ & & 0.8 & & & & \\ & & & 0.6 & 0.4 & & \\ 0.2 & 0.4 & 0.4 & & & & 1.0 \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{matrix}$$

Markov Chain with Rewards

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, $\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Chain with Rewards

Definition

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

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

Markov Chain with Rewards

Definition

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

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

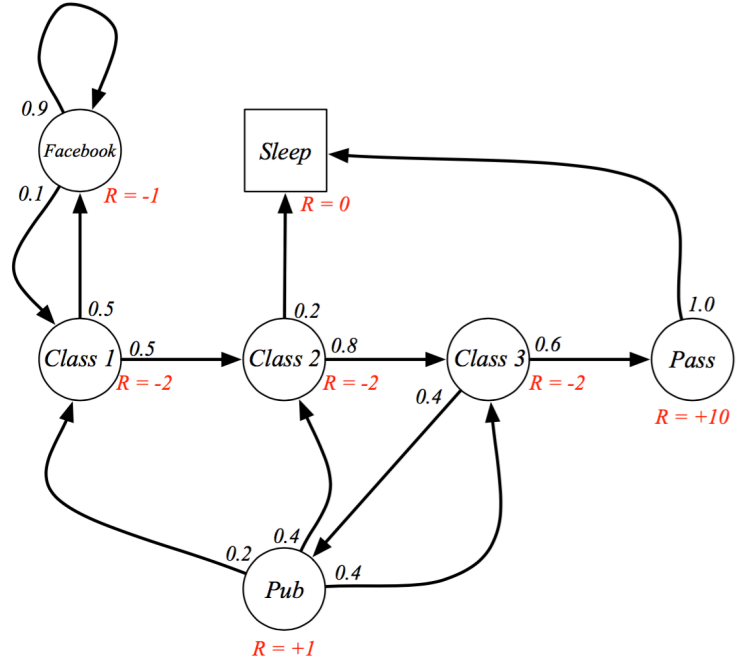
The value function $v(s)$ gives the long-term value of state s

Definition

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

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

Example Markov Reward Process



Sample **returns** for Student MRP:
 Starting from $S_1 = C1$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

¹Reference: David Silver, 2015

Recursions in Markov Reward Process

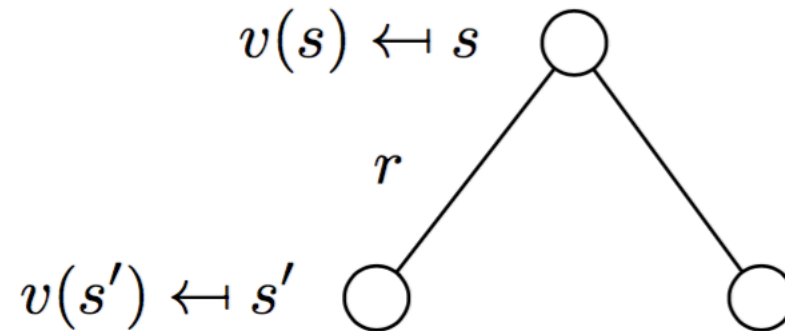
The value function can be decomposed into two parts:

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

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

Recursions in Markov Reward Process

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



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

Markov Decision Process

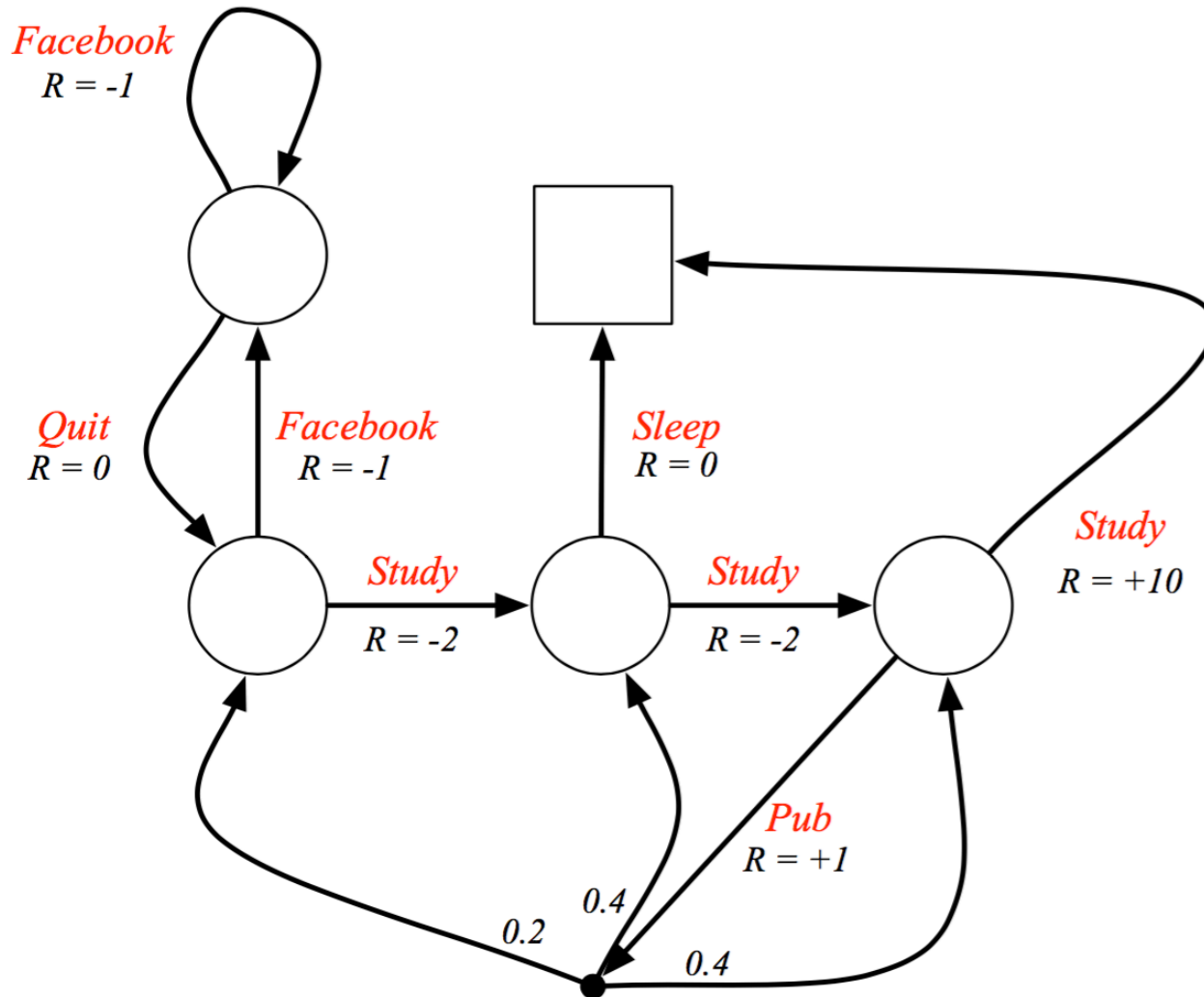
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

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, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.

Example Markov Decision Process



Markov Decision Process: Policy

- Now that we have introduced **actions**, we can discuss **policies** again
- Recall

Definition

A *policy* π 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 is an MRP for a Fixed Policy

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- 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 $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$

MDP is an MRP for a Fixed Policy

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- 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

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

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

Markov Decision Process: Value Function

- We can also talk about the **value function(s)**

Definition

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

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

Markov Decision Process: Value Function

- We can also talk about the value function(s)

Definition

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

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

Definition

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

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

Recursions in MDP

*Also called the [Bellman Expectation Equations](#)

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

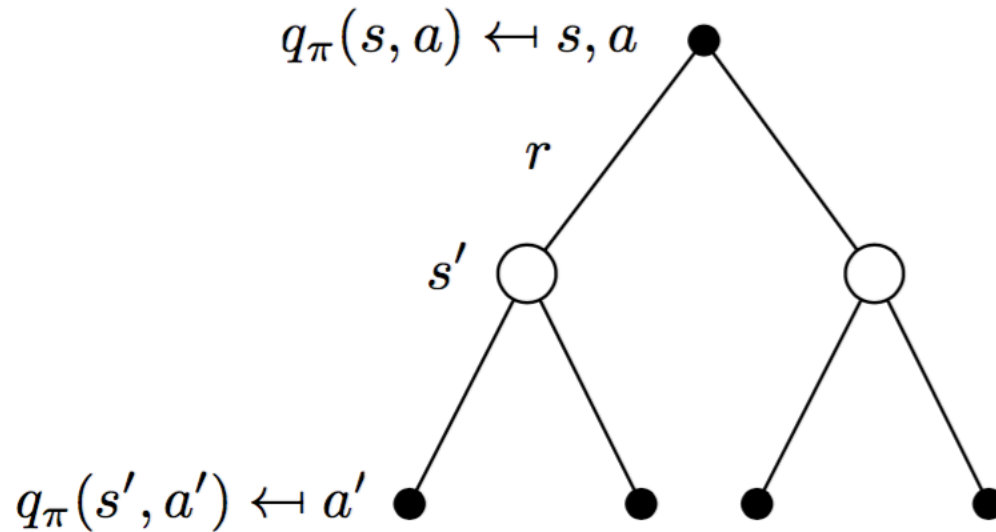
$$v_{\pi}(s) = \mathbb{E}_{\pi} [R_{t+1} + \gamma v_{\pi}(S_{t+1}) \mid S_t = s]$$

The action-value function can similarly be decomposed,

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

Recursions in MDP

*Also called the **Bellman Expectation Equations**



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

Markov Decision Process: Objective

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

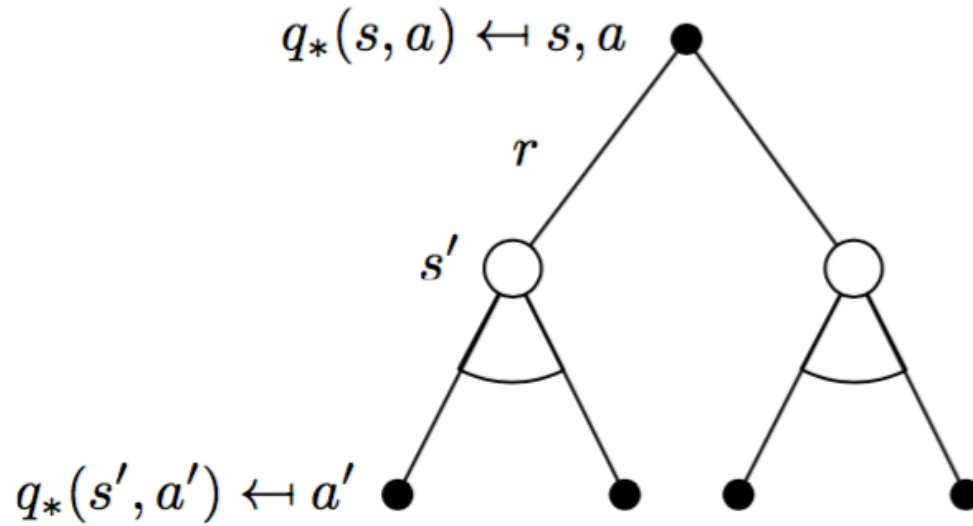
$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

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

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

Markov Decision Process: Objective

*Also called the **Bellman Optimality Equation**



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

Markov Decision Process: Optimal Policy

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

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

Questions?

Today's Outline

- Complex Decisions
- Reinforcement Learning Basics
 - Markov Decision Process
 - (State Action) Value Function
- Q Learning Algorithm

Finding the Best Policy

- Need to be able to do two things ideally
 - Prediction:
 - For a given policy, evaluate how good it is
 - Compute $q_{\pi}(s, a)$
 - Control:
 - And make an improvement from π
- We will focus on the Q Learning algorithm
 - It does prediction and control ‘simultaneously’

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1
- For $t = 1, 2, 3, \dots$
 - Take A_t chosen uniformly at random with probability ϵ

Explore

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1
- For $t = 1, 2, 3, \dots$
 - Take A_t chosen uniformly at random with probability ϵ
 - Take $\operatorname{argmax}_{a \in A} Q(S_t, a)$ with probability $1 - \epsilon$

Explore

Exploit

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1
- For $t = 1, 2, 3, \dots$
 - Take A_t chosen uniformly at random with probability ϵ Explore
 - Take $\operatorname{argmax}_{a \in A} Q(S_t, a)$ with probability $1 - \epsilon$ Exploit
 - Update Q :
 - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha_t \underbrace{(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t))}_{\text{Temporal difference error}}$

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1
- For $t = 1, 2, 3, \dots$
 - Take A_t chosen uniformly at random with probability ϵ Explore
 - Take $\operatorname{argmax}_{a \in A} Q(S_t, a)$ with probability $1 - \epsilon$ Exploit
 - Update Q :
 - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha_t \underbrace{(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t))}_{\text{Temporal difference error}}$
- Parameter ϵ is the exploration parameter
- Parameter α_t is the learning rate

The Q Learning Algorithm

- Initialize Q , which is a **table** of size $\#states \times \#actions$
- Start at state s_1
- For $t = 1, 2, 3, \dots$
 - Take A_t chosen uniformly at random with probability ϵ Explore
 - Take $\operatorname{argmax}_{a \in A} Q(S_t, a)$ with probability $1 - \epsilon$ Exploit
 - Update Q :
 - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha_t \underbrace{(R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a) - Q(S_t, A_t))}_{\text{Temporal difference error}}$
- Parameter ϵ is the exploration parameter
- Parameter α_t is the learning rate
- Under appropriate assumptions¹, $\lim_{t \rightarrow \infty} Q = Q^*$

¹Reference: Christopher J. C. H. Watkins and Peter Dayan, 1992

The Q Learning Algorithm: Notation

- Bellman Optimality Equation gives rise to the Q-Value Iteration algorithm
- Making this algorithm incremental, sampled and adding ϵ -greedy exploration gives Q Learning Algorithm

Q-Value Iteration

$$Q(s, a) \leftarrow \mathbb{E} \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a \right]$$

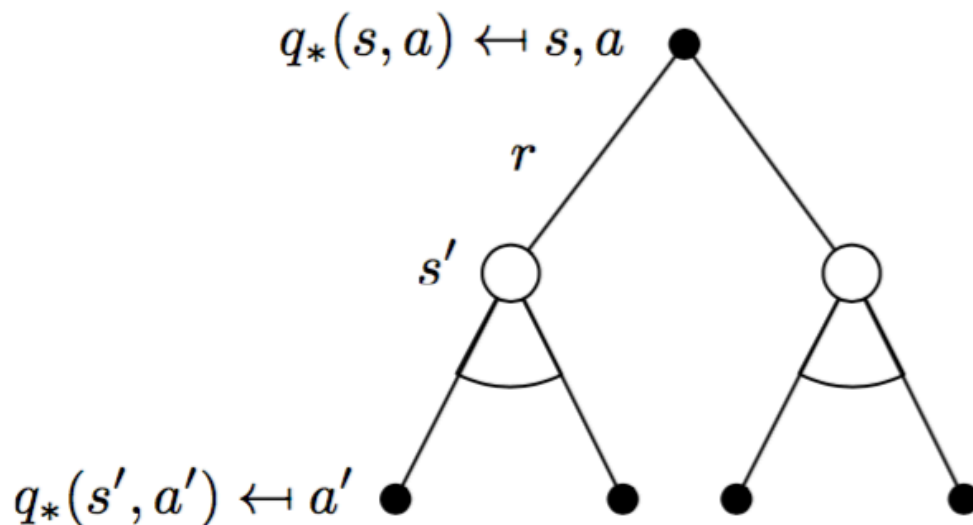
Q-Learning

$$Q(S, A) \stackrel{\alpha}{\leftarrow} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$$

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

Q-Value Iteration

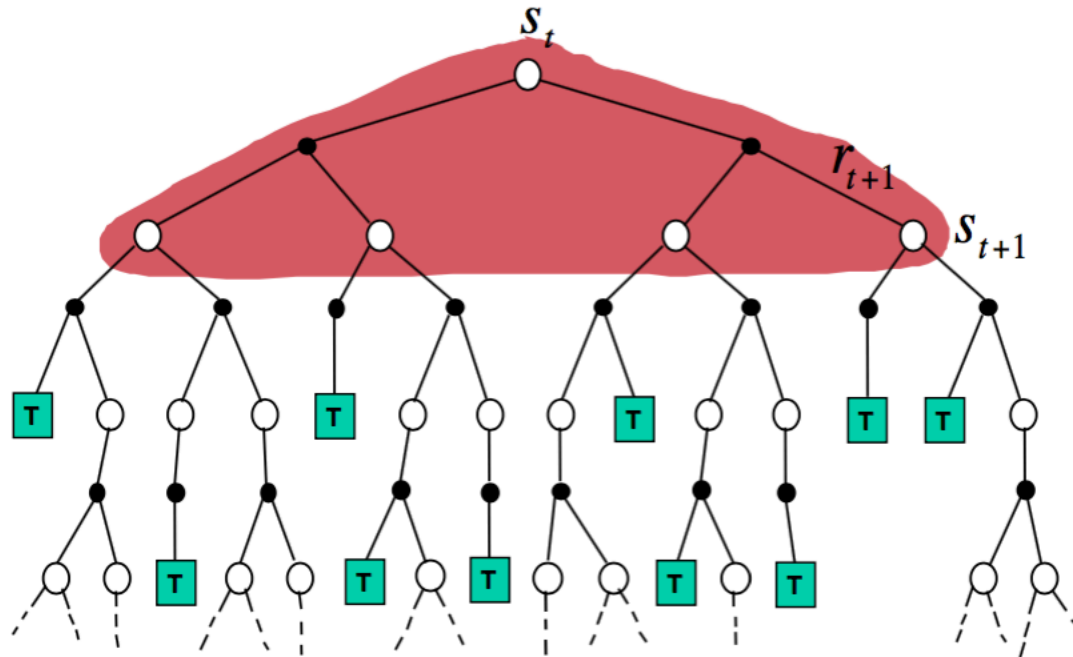
- If we know the model
 - Turn the Bellman Optimality Equation into an **iterative update**
 - Replace the equality with the 'arrow' sign below



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

Q-Value Iteration

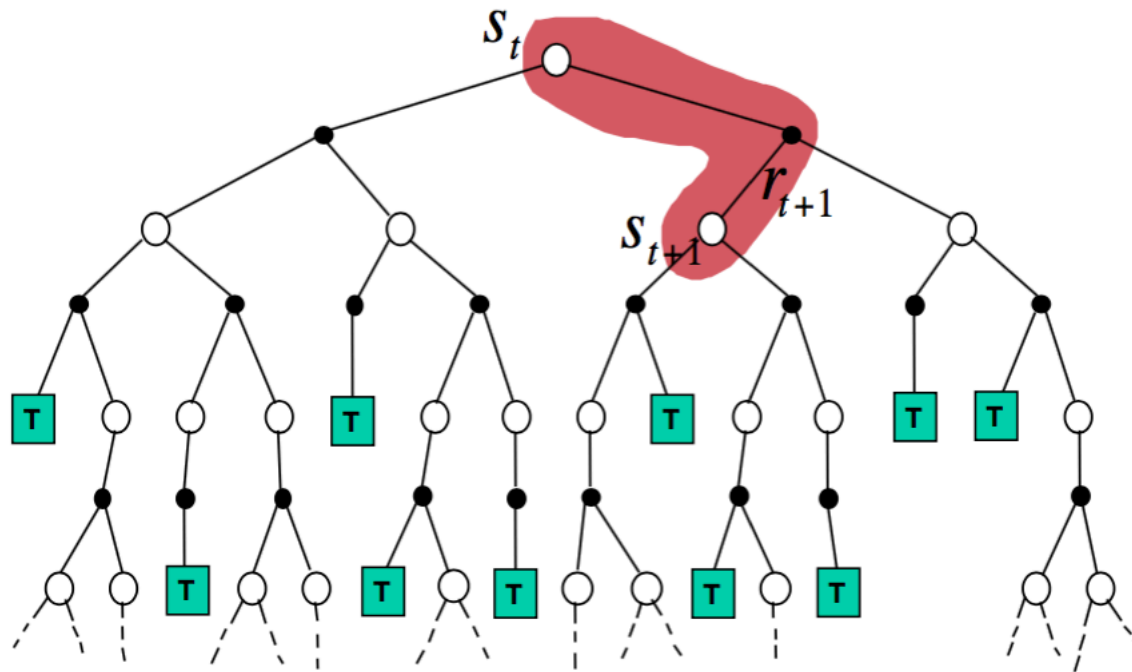
$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$



$$Q(s, a) \leftarrow \mathbb{E} \left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a \right]$$

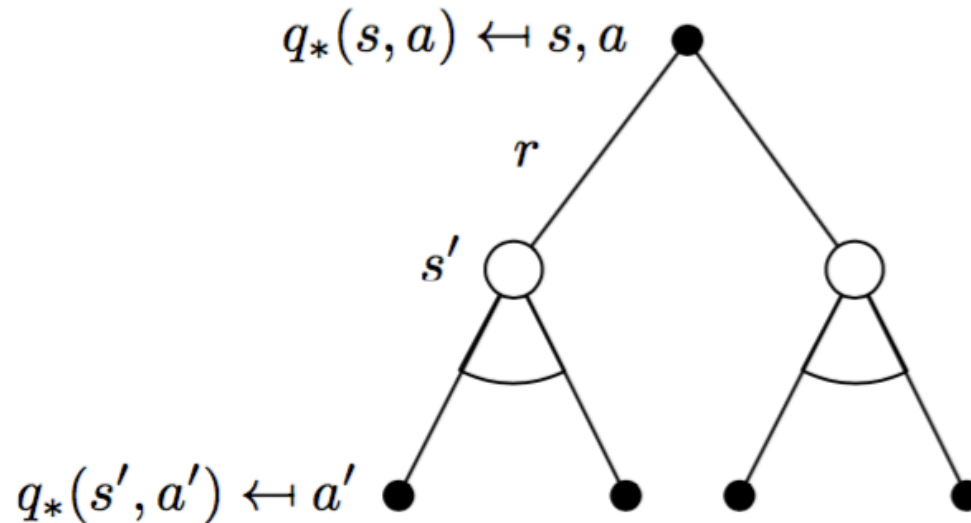
Q-Value Iteration

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



The Q Learning Algorithm

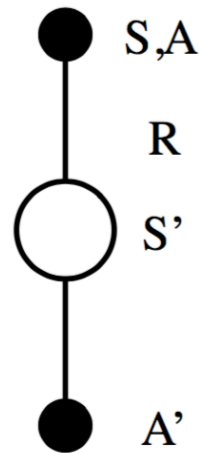
- If we do not know the model
 - Do **sampling** to get an **incremental** iterative update
 - Choose next actions to ensure **exploration**



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

The Q Learning Algorithm

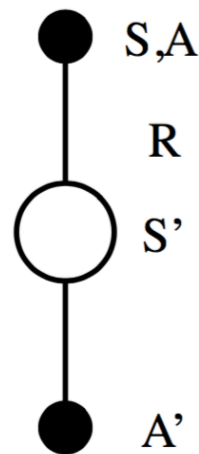
- If we do not know the model
 - Do **sampling** to get an **incremental** iterative update
 - Choose next actions to ensure **exploration**



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

The Q Learning Algorithm

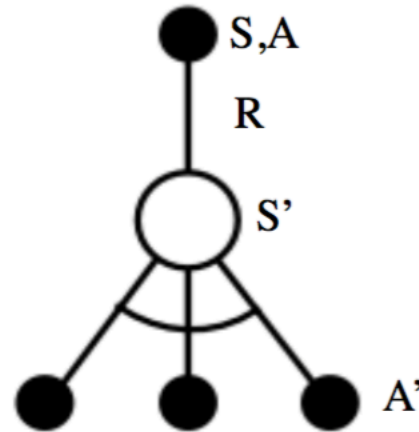
- If we do not know the model
 - Do **sampling** to get an **incremental** iterative update
 - Choose next actions to ensure **exploration**



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

The Q Learning Algorithm

- If we do not know the model
 - Do **sampling** to get an **incremental** iterative update
 - Choose next actions to ensure **exploration**



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

Questions?

Summary

- RL is a great framework to make agents intelligent
- Specify goals and provide feedback
- Many challenges still remain: exciting opportunity to contribute towards next generation of artificially intelligent and autonomous agents.
- In the next lecture, we will see that **deep learning function approximation based RL** agents show promise in large complex tasks: **representations** matter!
 - Applications such as
 - Self-driving cars
 - Intelligent virtual agents

Appendix

Sample Exam Questions

- What is the difference between a Markov Chain and a Markov Reward Process?
- What is the difference between a Markov Chain and a Markov Decision Process?
- Why is exploration needed in the reinforcement learning setting?
- What does the optimal state-action value function signify?
- What are the two objects (distributions) of an RL model?
- What is the difference between supervised learning and reinforcement learning?

Additional Resources

- An Introduction to Reinforcement Learning by Richard Sutton and Andrew Barto
 - <http://incompleteideas.net/sutton/book/the-book.html>
- Course on Reinforcement Learning by David Silver at UCL (includes video lectures)
 - <http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>
- Research Papers
 - Deep RL collection: <https://github.com/junhyukoh/deep-reinforcement-learning-papers>
 - [MKS RVBGR FOPBS AKKWLH2015] Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.
 - [SHMGSDSAPLDGNK SLLK GH2016] Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489, 2016.

Cons of RL

- Reinforcement Learning requires experiencing the environment many many times
- This is because it is a trial and error based approach

- Impractical for many complex tasks
- Unless one has access to simulators where an RL agent can practice a billion times

RL versus other Machine Learning Settings

- There is a notion of exploration and exploitation, similar to *Multi-armed bandits* and *Contextual bandits*
 - *Exploration* finds more information about the environment
 - *Exploitation* exploits known information to maximise reward
 - It is usually important to explore as well as exploit
- **Key difference: actions influence future contexts**
 - 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

RL versus other Sequential Decision Making Settings

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

Types of RL Agents

- There are many ways to design them, so we roughly categorize them as below:

- Value Based

- No Policy (Implicit)

- Value Function

- Policy Based

- Policy

- No Value Function

- Actor Critic

- Policy

- Value Function

- Model Free

- Policy and/or Value Function

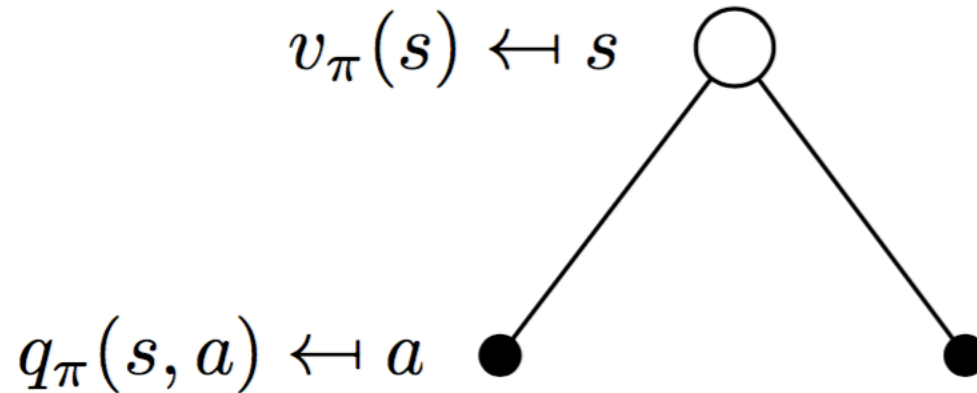
- No Model

- Model Based

- Policy and/or Value Function

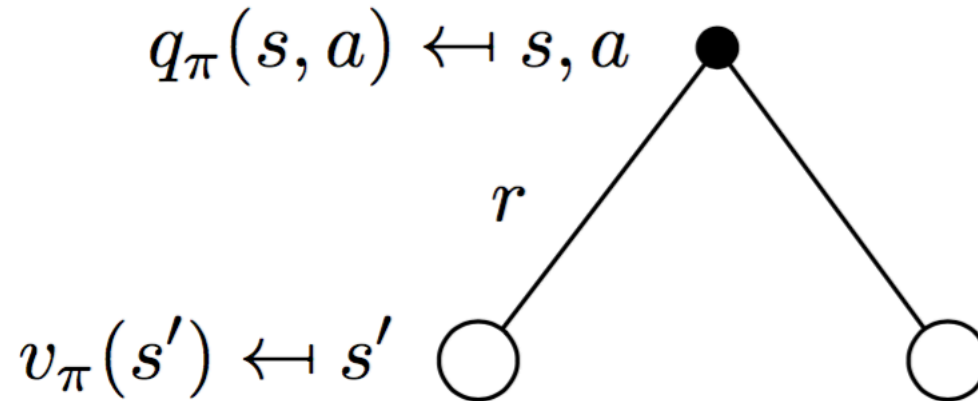
- Model

Relating the Two Value Functions I



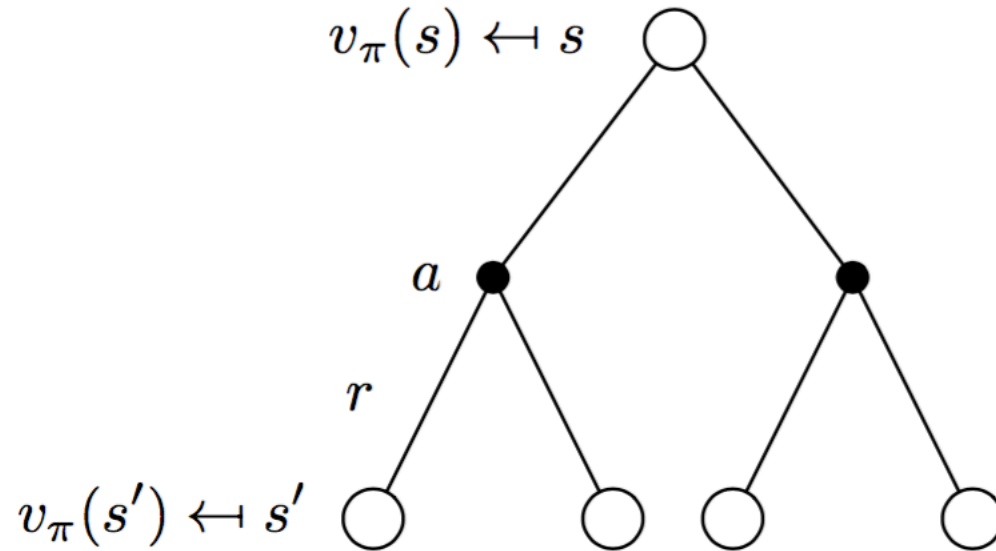
$$v_{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_{\pi}(s, a)$$

Relating the Two Value Functions II



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

Recursion in MDP: Value Function Version



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

Relating Policy and Value Function

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

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