

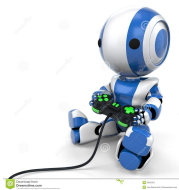
# STA414/2104

## Statistical Methods for Machine Learning II

Murat A. Erdogdu

Department of Computer Science  
Department of Statistical Sciences

Lecture 10



# Today

- Reinforcement learning: basics
- TA OHs for HW4 are on Thursday 1pm and Friday 1pm.

# Reinforcement Learning Problem

- In supervised learning, the problem is to predict an output  $t$  given an input  $x$ .
- But often the ultimate goal is not to predict, but to make decisions, i.e., take actions.
- And we need to take a sequence of actions.
- The actions have long-term consequences.



An agent



observes the world



takes an action and its states changes



with the goal of achieving long-term rewards.

**Reinforcement Learning Problem:** An agent continually interacts with the environment. How should it choose its actions so that its long-term rewards are maximized?

# Playing Games: Atari



<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

# Playing Games: Super Mario



[https://www.youtube.com/watch?v=wfL4L\\_14U9A](https://www.youtube.com/watch?v=wfL4L_14U9A)

# Making Pancakes!



[https://www.youtube.com/watch?v=W\\_gxLKSsSIE](https://www.youtube.com/watch?v=W_gxLKSsSIE)

# Reinforcement Learning Resources

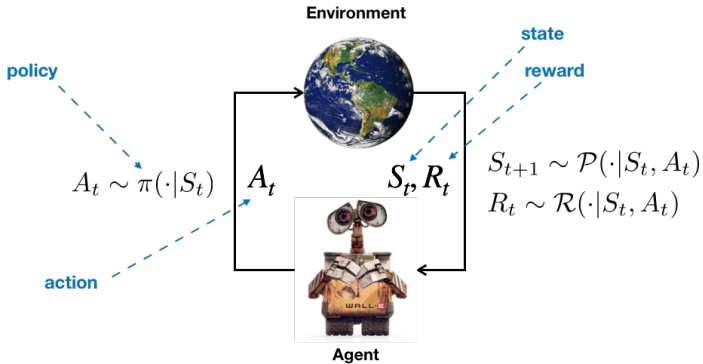
- *Reinforcement Learning: An Introduction second edition*, Sutton & Barto Book (2018)
- Video lectures by David Silver ([link](#)) .

# Reinforcement Learning

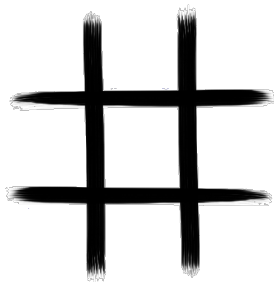
- Learning problems differ in the information available to the learner:
  - ▶ **Supervised:** For a given input, we know its corresponding output, e.g., class label
  - ▶ **Unsupervised:** We only have input data. We somehow need to organize them in a meaningful way, e.g., clustering.
  - ▶ **Reinforcement learning:** We observe inputs, and we have to choose outputs (actions) in order to maximize rewards. Correct outputs are not provided.
- In RL, we face the following challenges:
  - ▶ Continuous stream of input information, and we have to choose actions
  - ▶ Effects of an action depend on the state of the agent in the world
  - ▶ Obtain reward that depends on the state and actions
  - ▶ You know the reward for your action, not other possible actions.
  - ▶ Could be a delay between action and reward.



# Reinforcement Learning

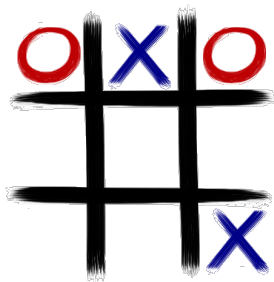


# Example: Tic Tac Toe, Notation



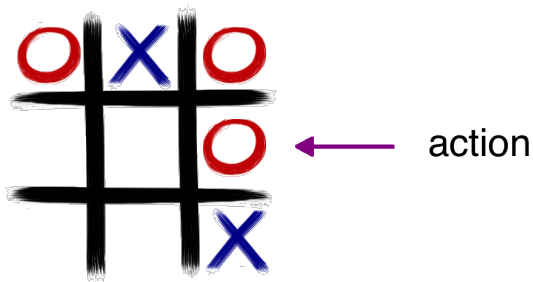
environment

# Example: Tic Tac Toe, Notation

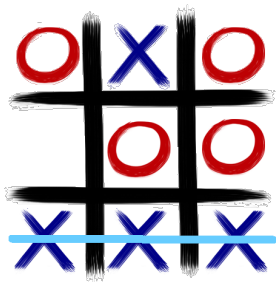


(current)  
state

# Example: Tic Tac Toe, Notation



# Example: Tic Tac Toe, Notation



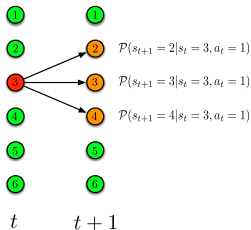
reward  
(here: -1)

# Formalizing Reinforcement Learning

- Markov Decision Process (MDP) is the mathematical framework to describe RL problems
- A discounted MDP is defined by a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ .
  - ▶  $\mathcal{S}$ : State space. Discrete or continuous
  - ▶  $\mathcal{A}$ : Action space. Here we consider finite action space, i.e.,  $\mathcal{A} = \{a_1, \dots, a_M\}$  if we have  $M = |\mathcal{A}|$  actions.
  - ▶  $\mathcal{P}$ : Transition probability
  - ▶  $\mathcal{R}$ : Immediate reward distribution
  - ▶  $\gamma$ : Discount factor ( $0 \leq \gamma < 1$ )
- Let us take a closer look at each of them.

# Formalizing Reinforcement Learning

- The agent has a **state**  $s \in \mathcal{S}$  in the environment, e.g., the location of X and O in tic-tac-toe, or the location of a robot in a room.
- At every time step  $t = 0, 1, \dots$ , the agent is at state  $s_t$ .
  - ▶ Takes an **action**  $a_t$
  - ▶ Moves into a new state  $s_{t+1}$ , according to the dynamics of the environment and the selected action, i.e.,  $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)$
  - ▶ Receives some **reward**  $R_t \sim \mathcal{R}(\cdot | s_t, a_t)$
  - ▶ Alternatively, it can be  $R_t \sim \mathcal{R}(\cdot | s_t)$  or ...



# Formulating Reinforcement Learning

- The action selection mechanism is described by a **policy**  $\pi$ 
  - ▶ Policy  $\pi$  is a mapping from states to actions, i.e.,  $a_t = \pi(s_t)$  (deterministic policy) or  $A_t \sim \pi(\cdot|s_t)$  (stochastic policy).
- The goal is to find a policy  $\pi$  such that **long-term rewards** of the agent is maximized.
- Different notions of the long-term reward:
  - ▶ Cumulative/total reward:  $r_0 + r_1 + r_2 + \dots$
  - ▶ Discounted (cumulative) reward:  $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ 
    - ▶ The discount factor  $0 \leq \gamma \leq 1$  determines how myopic or farsighted the agent is.
    - ▶ When  $\gamma$  is closer to **0**, the agent prefers to obtain reward as soon as possible.
    - ▶ When  $\gamma$  is close to **1**, the agent is willing to receive rewards in the farther future.
    - ▶ The discount factor  $\gamma$  has a financial interpretation: If a dollar next year is worth almost the same as a dollar today,  $\gamma$  is close to **1**. If a dollar's worth next year is much less its worth today,  $\gamma$  is close to **0**.

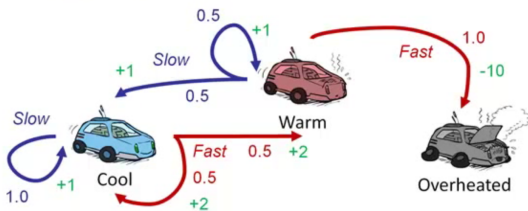


# Transition Probability (or Dynamics)

- The transition probability describes the changes in the state of the agent when it chooses actions

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

- This model has **Markov property**: the future depends on the past only through the current state



# Policy

- A **policy** is the action selection mechanism of the agent, and describes its behaviour.
- Policy can be deterministic or stochastic:
  - ▶ Deterministic policy:  $a = \pi(s)$
  - ▶ Stochastic policy:  $A \sim \pi(\cdot|s)$

# Value Function

- **Value function** is the expected future reward, and is used to evaluate the desirability of states.
- State-value function  $V^\pi$  (or simply value function) for policy  $\pi$  is a function defined as

$$V^\pi(s) \triangleq \mathbb{E}_\pi \left[ \sum_{t \geq 0} \gamma^t R_t \mid S_0 = s \right] \quad \text{where } R_t \sim \mathcal{R}(\cdot | S_t, A_t).$$

It describes the expected discounted reward if the agent starts from state  $s$  and follows policy  $\pi$ .

- The action-value function  $Q^\pi$  for policy  $\pi$  is

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi \left[ \sum_{t \geq 0} \gamma^t R_t \mid S_0 = s, A_0 = a \right].$$

It describes the expected discounted reward if the agent starts from state  $s$ , takes action  $a$ , and afterwards follows policy  $\pi$ .

# Value Function

- The goal is to find a policy  $\pi$  that maximizes the value function
- Optimal value function:

$$Q^*(s, a) = \sup_{\pi} Q^{\pi}(s, a)$$

- Given  $Q^*$ , the optimal policy can be obtained as

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

- The goal of an RL agent is to find a policy  $\pi$  that is close to optimal, i.e.,  $Q^{\pi} \approx Q^*$ .

# Example: Tic-Tac-Toe

- Consider the game tic-tac-toe:
  - ▶ State: Positions of X's and O's on the board
  - ▶ Action: The location of the new X or O.
  - ▶ Policy: mapping from states to actions
  - ▶ Reward: win/lose/tie the game ( $+1/ - 1/0$ ) [only at final move in given game]
    - ▶ based on rules of game: choice of one open position
  - ▶ Value function: Prediction of reward in future, based on current state
- In tic-tac-toe, since state space is tractable, we can use a table to represent value function

# RL & Tic-Tac-Toe

- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - ▶ start with all values = 0.5
  - ▶ **policy**: choose move with highest probability of winning given current legal moves from current state
  - ▶ update entries in table based on outcome of each game
  - ▶ After many games value function will represent true probability of winning from each state
- Can try alternative policy: sometimes select moves randomly (exploration)

# Bellman Equation

- Recall that reward at time  $t$  is a random variable  $R_t$ . In the sequel, we will assume that reward depends only on the current state and the action  $R_t \sim \mathcal{R}(\cdot|S_t, A_t)$ .

We can write it as  $R_t = R(S_t, A_t)$  for a random function  $R$  with expected value  $r(s, a) = \mathbb{E}[R(s, a)]$ .

- The value function satisfies the following recursive relationship:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a \right] \\ &= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t R(S_t, A_t) | S_0 = s, A_0 = a \right] \\ &= \mathbb{E} [R(S_0, A_0) + \gamma R(S_1, A_1) + \gamma^2 R(S_2, A_2) + \dots | S_0 = s, A_0 = a] \\ &= \mathbb{E} [R(s, a) + \gamma \{R(S_1, \pi(S_1)) + \gamma R(S_2, \pi(S_2)) + \dots\} | S_0 = s, A_0 = a] \\ &= \mathbb{E} [R(s, a) + \gamma Q^\pi(S_1, \pi(S_1)) | S_0 = s, A_0 = a] \\ &= \underbrace{r(s, a) + \gamma \int_{\mathcal{S}} Q^\pi(s', \pi(s')) \mathcal{P}(s' | s, a) ds'}_{\triangleq (T^\pi Q^\pi)(s, a)} \end{aligned}$$

# Bellman Equation

The value function satisfies

$$Q^\pi(s, a) = r(s, a) + \underbrace{\gamma \int_{\mathcal{S}} Q^\pi(s', \pi(s')) \mathcal{P}(s'|s, a) ds'}_{\triangleq (T^\pi Q^\pi)(s, a)}$$

$$(T^\theta Q)(s, a) \triangleq r(s, a) + \gamma \int_{\mathcal{S}} Q(s', \theta(s')) \mathcal{P}(s'|s, a) ds'$$

This is called the **Bellman equation** and  $T^\pi$  is the **Bellman operator**.  
Similarly, we define the **Bellman optimality operator**  $T^*$ :

$$(T^* Q)(s, a) \triangleq r(s, a) + \gamma \int_{\mathcal{S}} \max_{a' \in \mathcal{A}} Q(s', a') \mathcal{P}(s'|s, a) ds'$$



# Bellman Equation

- Key observation for  $Q^*(s, a) = \sup_{\pi} Q^{\pi}(s, a)$  and  $\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$ : We have

$$Q^{\pi} = T^{\pi} Q^{\pi}$$

$$Q^* = T^* Q^*$$

- The solution of these fixed-point equations are unique.
- Value-based approaches try to find a  $\hat{Q}$  such that

$$\hat{Q} \approx T^* \hat{Q}$$

- The **greedy policy** of  $\hat{Q}$  is close to the optimal policy:

$$Q^{\pi(s; \hat{Q})} \approx Q^{\pi^*} = Q^*$$

where the greedy policy of  $\hat{Q}$  is defined as

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

# Finding the Value Function

- Let us first study the **policy evaluation** problem: Given a policy  $\pi$ , find  $Q^\pi$  (or  $V^\pi$ ).
- Policy evaluation is an intermediate step for many RL methods.
- The uniqueness of the fixed-point of the Bellman operator implies that if we find a  $Q$  such that  $T^\pi Q = Q$ , then  $Q = Q^\pi$ .
- Assume that  $\mathcal{P}$  and  $\mathcal{R}$  are known (big assumption!).
- If the state-action space  $\mathcal{S} \times \mathcal{A}$  is finite (and not very large, i.e., hundreds or thousands, but not millions or billions), we can solve the following Linear System of Equations over  $Q$ :

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) Q(s', \pi(s')) \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A}$$

- This is feasible for small problems ( $|\mathcal{S} \times \mathcal{A}|$  is not too large), but for large problems there are better approaches.

# Finding the Optimal Value Function

- The Bellman optimality operator also has a unique fixed point.
- If we find a  $Q$  such that  $T^*Q = Q$ , then  $Q = Q^*$ .
- Let us try an approach similar to what we did for the policy evaluation problem.
- If the state-action space  $\mathcal{S} \times \mathcal{A}$  is finite (and not very large), we can solve the following Nonlinear System of Equation:

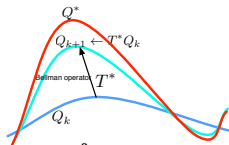
$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a') \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A}$$

- This is a nonlinear system of equations, and can be difficult to solve. Can we do anything else?

# Value Iteration

- Assume that we know the model  $\mathcal{P}$  and  $\mathcal{R}$ . How can we find the optimal value function?
- Finding the optimal policy/value function when the **model is known** is sometimes called the **planning** problem.
- We can benefit from the Bellman optimality equation and use a method called **Value Iteration**: Start from an initial function  $Q_1$ . For each  $k = 1, 2, \dots$ , apply

$$Q_{k+1} \leftarrow T^* Q_k$$

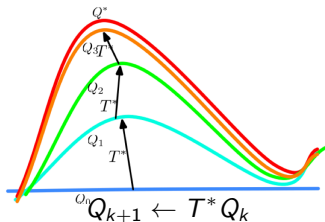


$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \int_{\mathcal{S}} \max_{a' \in \mathcal{A}} Q_k(s', a') \mathcal{P}(s' | s, a) ds'$$

$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \max_{a' \in \mathcal{A}} Q_k(s', a') \mathcal{P}(s' | s, a)$$

# Value Iteration

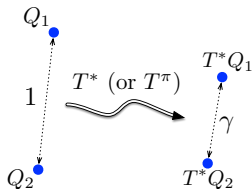
- The Value Iteration converges to the optimal value function.
- This is because of the contraction property of the Bellman (optimality) operator, i.e.,  $\|T^* Q_1 - T^* Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$ .



$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \int_{\mathcal{S}} \mathcal{P}(ds' | s, a) \max_{a' \in \mathcal{A}} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s' | s, a) \max_{a' \in \mathcal{A}} Q_k(s', a')$$

# Bellman Operator is Contraction



$$\begin{aligned} |(T^* Q_1)(s, a) - (T^* Q_2)(s, a)| &= \left| \left[ r(s, a) + \gamma \int_S \mathcal{P}(ds' | s, a) \max_{a' \in \mathcal{A}} Q_1(s', a') \right] - \right. \\ &\quad \left. \left[ r(s, a) + \gamma \int_S \mathcal{P}(ds' | s, a) \max_{a' \in \mathcal{A}} Q_2(s', a') \right] \right| \\ &= \gamma \left| \int_S \mathcal{P}(ds' | s, a) \left[ \max_{a' \in \mathcal{A}} Q_1(s', a') - \max_{a' \in \mathcal{A}} Q_2(s', a') \right] \right| \\ &\leq \gamma \int_S \mathcal{P}(ds' | s, a) \max_{a' \in \mathcal{A}} |Q_1(s', a') - Q_2(s', a')| \\ &\leq \gamma \max_{(s', a') \in \mathcal{S} \times \mathcal{A}} |Q_1(s', a') - Q_2(s', a')| \underbrace{\int_S \mathcal{P}(ds' | s, a)}_{=1} \end{aligned}$$

# Bellman Operator is Contraction

Therefore, we get that

$$\sup_{(s,a) \in \mathcal{S} \times \mathcal{A}} |(T^* Q_1)(s, a) - (T^* Q_2)(s, a)| \leq \gamma \sup_{(s,a) \in \mathcal{S} \times \mathcal{A}} |Q_1(s, a) - Q_2(s, a)|.$$

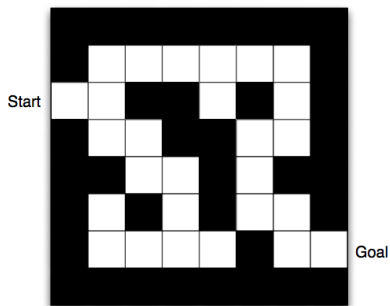
Or more succinctly,

$$\|T^* Q_1 - T^* Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty.$$

We also have a similar result for the Bellman operator of a policy  $\pi$ :

$$\|T^\pi Q_1 - T^\pi Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty.$$

# Maze Example

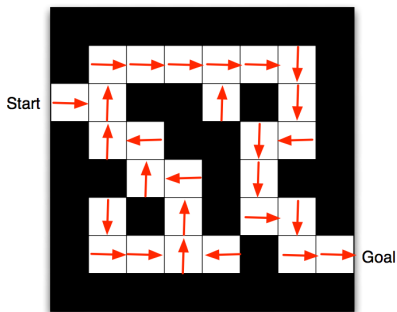


- Rewards:  $-1$  per time-step
- Actions: N, E, S, W
- States: Agent's location

[Slide credit: D. Silver]



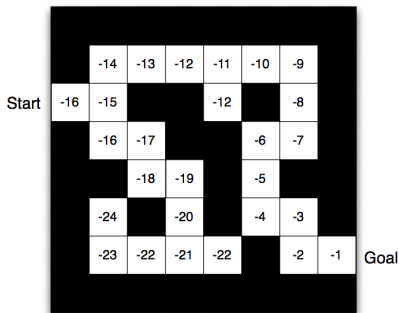
# Maze Example



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

[Slide credit: D. Silver]

# Maze Example



- Numbers represent value  $V^\pi(s)$  of each state  $s$

[Slide credit: D. Silver]

# Challenges

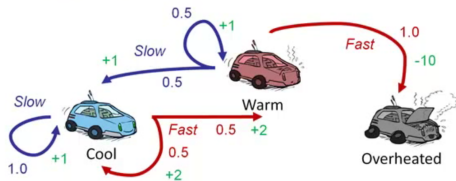
- When we have a large state space (e.g., when  $\mathcal{S} \subset \mathbb{R}^d$  or  $|\mathcal{S} \times \mathcal{A}|$  is very large):
  - ▶ Exact representation of the value ( $Q$ ) function is infeasible for all  $(s, a) \in \mathcal{S} \times \mathcal{A}$ .
  - ▶ The exact integration in the Bellman operator is challenging

$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \int_{\mathcal{S}} \max_{a' \in \mathcal{A}} Q_k(s', a') \mathcal{P}(s' | s, a) ds'$$

- We often do not know the dynamics  $\mathcal{P}$  and the reward function  $\mathcal{R}$ , so we cannot calculate the Bellman operators.

# Recall MDP

- Markov Decision Problem (MDP): tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ .
- Main assumption: Markovian dynamics and reward.
- Standard MDP problems:
  1. **Planning:** given complete Markov decision problem as input, compute policy with optimal expected return

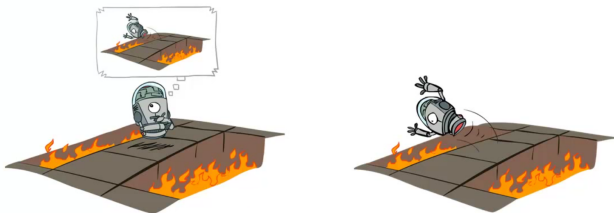


[Pic: P. Abbeel]

# Basic Problems

- Standard MDP problems:
  1. **Planning:** given complete Markov decision problem as input, compute policy with optimal expected return
  2. **Learning:** We don't know which states are good or what the actions do. We must try out the actions and states to learn what to do

# Example of Standard MDP Problem

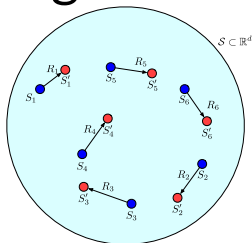


1. **Planning:** given complete Markov decision problem as input, compute policy with optimal expected return
2. **Learning:** Only have access to experience in the MDP, learn a near-optimal strategy

# Is There Any Hope?

- During this course, we saw many methods to learn functions (e.g., classifier, regressor) when the input is continuous-valued and we are only given a finite number of data points.
- We may adopt those techniques to solve RL problems.
- There are some other aspects of the RL problem that we do not touch in this course; we briefly mention them later.

# Batch RL and Approximate Dynamic Programming



- Suppose that we are given the following dataset

$$\mathcal{D}_N = \{(S_i, A_i, R_i, S'_i)\}_{i=1}^N$$

$$(S_i, A_i) \sim \nu \quad (\nu \text{ is a distribution over } \mathcal{S} \times \mathcal{A})$$

$$S'_i \sim \mathcal{P}(\cdot | S_i, A_i)$$

$$R_i \sim \mathcal{R}(\cdot | S_i, A_i)$$

- Can we estimate  $Q \approx Q^*$  using these data?



# Approximate Value Iteration

- Recall that each iteration of VI computes

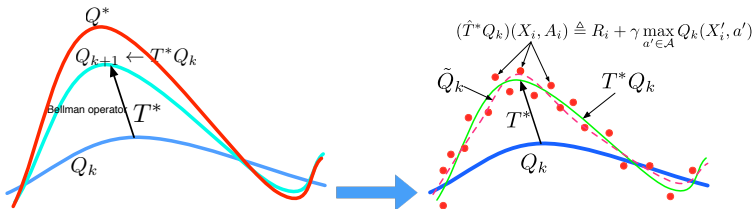
$$Q_{k+1} \leftarrow T^* Q_k$$

- We cannot directly compute  $T^* Q_k$ . But we can use data to approximately perform one step of VI.
- Consider  $(S_i, A_i, R_i, S'_i)$  from the dataset  $\mathcal{D}_N$ .
- Consider a function  $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ .
- We can define a random variable  $t_i = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')$ .
- Notice that

$$\begin{aligned} \mathbb{E}[t_i | S_i, A_i] &= \mathbb{E} \left[ R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a') | S_i, A_i \right] = \\ & r(S_i, A_i) + \gamma \int \max_{a' \in \mathcal{A}} Q(s', a') \mathcal{P}(s' | S_i, A_i) ds' = (T^* Q)(S_i, A_i) \end{aligned}$$

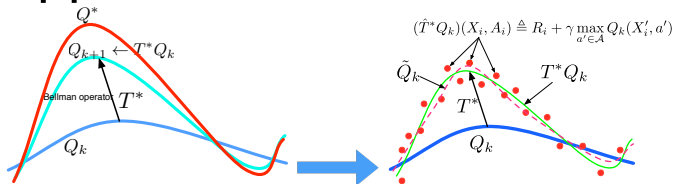
- So  $t_i = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')$  is a noisy version of  $(T^* Q)(S_i, A_i)$ . Fitting a function to noisy real-valued data is the regression problem.

# From Value Iteration to Approximate Value Iteration



- Given the dataset  $\mathcal{D}_N = \{(S_i, A_i, R_i, S'_i)\}_{i=1}^N$  and an action-value function estimate  $Q_k$ , we can construct the dataset  $\{(x_i, t_i)\}_{i=1}^N$  with  $x_i = (S_i, A_i)$  and  $t_i = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')$ .
- Because of  $\mathbb{E}[R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(S'_i, a') | S_i, A_i] = (T^* Q_k)(S_i, A_i)$  we can treat the problem of estimating  $Q_{k+1}$  as a regression problem with noisy data.

# Approximate Value Iteration



- Given the dataset  $\mathcal{D}_N = \{(S_i, A_i, R_i, S'_i)\}_{i=1}^N$  and an action-value function estimate  $Q_k$ , we solve a regression problem. We minimize the squared error:

$$Q_{k+1} \leftarrow \operatorname{argmin}_{Q \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N \left| Q(S_i, A_i) - \underbrace{\left( R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(S'_i, a') \right)}_{t_i} \right|^2$$

- We run this procedure  $K$ -times.
- The policy of the agent is selected to be the **greedy** policy w.r.t. the final estimate of the value function: At state  $s \in \mathcal{S}$ , the agent chooses  $\pi(s; Q_K) \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} Q_K(s, a)$
- This method is called **Approximate Value Iteration** or **Fitted Value Iteration**.

# Choice of Estimator

We have many choices for the regression method (and the function space  $\mathcal{F}$ ):

- Linear models:  $\mathcal{F} = \{Q(s, a) = \mathbf{w}^\top \boldsymbol{\psi}(s, a)\}$ .
  - ▶ How to choose the feature mapping  $\boldsymbol{\psi}$ ?
- Decision Trees, Random Forest, etc.
- Kernel-based methods, and regularized variants.
- (Deep) Neural Networks. Deep Q Network (DQN) is an example of performing AVI with DNN, with some DNN-specific tweaks.

# Some Remarks on AVI

- AVI converts a value function estimation problem to a sequence of regression problems.
- As opposed to the conventional regression problem, the target of AVI, which is  $T^*Q_k$ , changes at each iteration.
- Usually we cannot guarantee that the solution of the regression problem  $Q_{k+1}$  is exactly equal to  $T^*Q_k$ . We may only have  $Q_{k+1} \approx T^*Q_k$ .
- These errors might accumulate and may even cause divergence.

# From Batch RL to Online RL

- We started from the setting where the model was known (Planning) to the setting where we do not know the model, but we have a batch of data coming from the previous interaction of the agent with the environment (Batch RL).
- This allowed us to use tools from the supervised learning literature (particularly, regression) to design RL algorithms.
- But RL problems are often interactive: the agent continually interacts with the environment, updates its knowledge of the world and its policy, with the goal of achieving as much rewards as possible.
- Can we obtain an online algorithm for updating the value function?
- An extra difficulty is that an RL agent should handle its interaction with the environment carefully: it should collect as much information about the environment as possible (**exploration**), while benefitting from the knowledge that has been gathered so far in order to obtain a lot of rewards (**exploitation**).

# Online RL

- Suppose that agent continually interacts with the environment. This means that
  - ▶ At time step  $t$ , the agent observes the state variable  $S_t$ .
  - ▶ The agent chooses an action  $A_t$  according to its policy, i.e.,  $A_t = \pi_t(S_t)$ .
  - ▶ The state of the agent in the environment changes according to the dynamics. At time step  $t + 1$ , the state is  $S_{t+1} \sim \mathcal{P}(\cdot | S_t, A_t)$ . The agent observes the reward variable too:  $R_t \sim \mathcal{R}(\cdot | S_t, A_t)$ .
- Two questions:
  - ▶ Can we update the estimate of the action-value function  $Q$  online and only based on  $(S_t, A_t, R_t, S_{t+1})$  such that it converges to the optimal value function  $Q^*$ ?
  - ▶ What should the policy  $\pi_t$  be so the algorithm explores the environment?
- **Q-Learning** is an online algorithm that addresses these questions.
- We present Q-Learning for finite state-action problems.

# Q-Learning with $\epsilon$ -Greedy Policy

- Parameters:
  - ▶ Learning rate:  $0 < \alpha < 1$ : learning rate
  - ▶ Exploration parameter:  $\epsilon$
- Initialize  $Q(s, a)$  for all  $(s, a) \in \mathcal{S} \times \mathcal{A}$
- The agent starts at state  $S_0$ .
- For time step  $t = 0, 1, \dots$ ,
  - ▶ Choose  $A_t$  according to the  **$\epsilon$ -greedy policy**, i.e.,

$$A_t \leftarrow \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}} Q(S_t, a) & \text{with probability } 1 - \epsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \epsilon \end{cases}$$

- ▶ Take action  $A_t$  in the environment.
- ▶ The state of the agent changes from  $S_t$  to  $S_{t+1} \sim \mathcal{P}(\cdot | S_t, A_t)$
- ▶ Observe  $S_{t+1}$  and  $R_t$
- ▶ Update the action-value function at state-action  $(S_t, A_t)$ :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_t + \gamma \max_{a' \in \mathcal{A}} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$



# Exploration vs. Exploitation

- The  $\epsilon$ -greedy is a simple mechanism for maintaining **exploration-exploitation tradeoff**.

$$\pi_{\epsilon}(S; Q) = \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}} Q(S, a) & \text{with probability } 1 - \epsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \epsilon \end{cases}$$

- The  $\epsilon$ -greedy policy ensures that most of the time (probability  $1 - \epsilon$ ) the agent exploits its incomplete knowledge of the world by chooses the best action (i.e., corresponding to the highest action-value), but occasionally (probability  $\epsilon$ ) it explores other actions.
- Without exploration, the agent may never find some good actions.
- The  $\epsilon$ -greedy is one of the simplest, but widely used, methods for trading-off exploration and exploitation. Exploration-exploitation tradeoff is an important topic of research.

# Softmax Action Selection

- We can also choose actions by sampling from probabilities derived from softmax function.

$$\pi_{\beta}(S; Q) = A \sim \mathbb{P}(A = a) = \frac{\exp\{\beta Q(S, a)\}}{\sum_{a \in \mathcal{A}} \exp\{\beta Q(S, a)\}}$$

- $\beta \rightarrow \infty$  recovers greedy action selection.

# Examples of Exploration-Exploitation in the Real World

- 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

[Slide credit: D. Silver]

# Intuition on Why Q-Learning Works?

- Consider a tuple  $(S, A, R, S')$ . The Q-learning update is

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

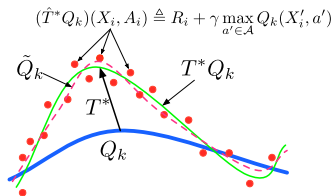
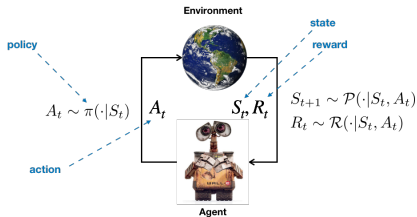
- To understand this better, let us focus on its stochastic equilibrium, i.e., where the expected change in  $Q(S, A)$  is zero. We have

$$\begin{aligned} \mathbb{E} \left[ R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S, A) | S, A \right] &= 0 \\ \Rightarrow (T^* Q)(S, A) &= Q(S, A) \end{aligned}$$

- So at the stochastic equilibrium, we have  $(T^* Q)(S, A) = Q(S, A)$ . Because the fixed-point of the Bellman optimality operator is unique (and is  $Q^*$ ),  $Q$  is the same as the optimal action-value function  $Q^*$ .
- One can show that under certain conditions, Q-Learning indeed converges to the optimal action-value function  $Q^*$  for finite state-action spaces.

# Recap and Other Approaches

- We defined MDP as the mathematical framework to study RL problems.
- We started from the assumption that the model is known (Planning). We then relaxed it to the assumption that we have a batch of data (Batch RL). Finally we briefly discussed Q-learning as an online algorithm to solve RL problems (Online RL).



# Function approximation

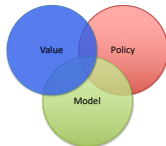
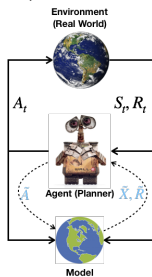
- Q-learning still scales badly with large state spaces, how many states does a chess game have? Need to save the full table!
- Similar states, e.g. move all chess pieces two steps to the left, treated as totally different.
- Solution: Instead of  $Q$  being a  $S \times A$  table it is a parametrized function.
- Looking for function  $\hat{Q}(s, a; \mathbf{w}) \approx Q^*(s, a)$ 
  - ▶ Linear functions  $Q(s, a; \mathbf{w}) = \mathbf{w}^T \phi(s, a)$ .
  - ▶ Neural network
- Hopefully can generalize to unseen states.
- Problem: Each change to parameters changes all states/actions - can lead to instability.
- Due to non-linearity, Q-learning can diverge.

# RL recap

- Learning from experience not from labeled examples.
- Why is RL hard?
  - ▶ Limited feedback.
  - ▶ Delayed rewards.
  - ▶ Huge state space.
- Usually solved by learning the value function or optimizing the policy (not covered)
- How do you define the rewards? Can be tricky.
  - ▶ Bad rewards can lead to **reward hacking**

# Recap and Other Approaches

- All discussed approaches estimate the value function first. They are called **value-based methods**.
- There are methods that directly optimize the policy, i.e., **policy search methods**.
- **Model-based RL** methods estimate the true, but unknown, model of environment  $\mathcal{P}$  by an estimate  $\hat{\mathcal{P}}$ , and use the estimator  $\hat{\mathcal{P}}$  in order to plan.
- There are hybrid methods.





# Reinforcement Learning Resources

- Books:
  - ▶ Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, 2nd edition, 2018.
  - ▶ Csaba Szepesvari, Algorithms for Reinforcement Learning, 2010.
  - ▶ Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, 2010.
  - ▶ Dimitri P. Bertsekas and John N. Tsitsiklis, Neuro-Dynamic Programming, 1996.
- Courses:
  - ▶ Video lectures by David Silver
  - ▶ CIFAR and Vector Institute's Reinforcement Learning Summer Schools, 2018+.
  - ▶ Deep Reinforcement Learning, CS 294-112 at UC Berkeley