

Introduction to Machine Learning

Reinforcement Learning

Varun Chandola

Computer Science & Engineering
State University of New York at Buffalo
Buffalo, NY, USA
chandola@buffalo.edu



Outline

Introduction to Reinforcement Learning Tic-Tac-Toe Example

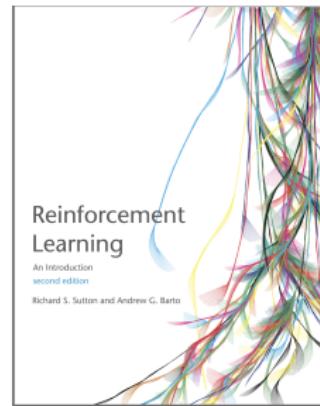
Markov Decision Processes

Introduction

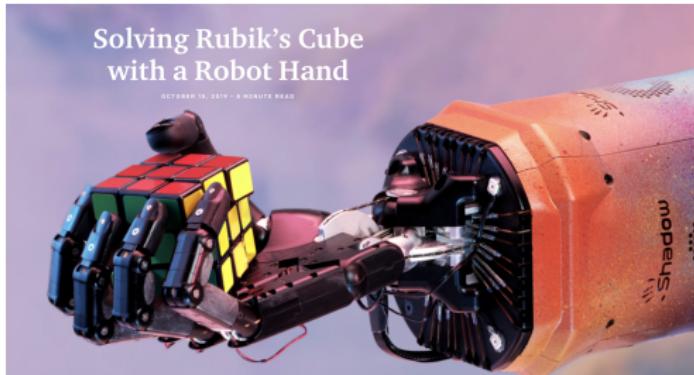
Special Thanks to Alina Vereschaka

- ▶ CSE410/510 - Introduction to Reinforcement Learning

- ▶ Reinforcement Learning -Sutton and Barto



What is Reinforcement Learning?



<https://www.youtube.com/watch?v=x408pojMF0w>

- ▶ Learn to take **actions** over a sequence of steps, to maximize **reward** over many time steps.

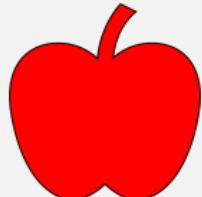
Comparing all ML problems

Supervised Learning

Data: $\langle \mathbf{x}_i, y_i \rangle$

Task: Infer y^* for \mathbf{x}^*

Example:



This is an apple

Comparing all ML problems

Supervised Learning

Data: $\langle \mathbf{x}_i, y_i \rangle$

Task: Infer y^* for \mathbf{x}^*

Example:



This is an apple

Unsupervised Learning

Data: $\langle \mathbf{x}_i \rangle$

Task: Learn **structure**

Example:



There are two types of apples

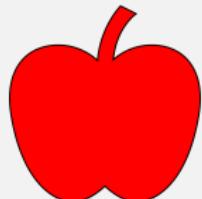
Comparing all ML problems

Supervised Learning

Data: $\langle \mathbf{x}_i, y_i \rangle$

Task: Infer y^* for \mathbf{x}^*

Example:



This is an apple

Unsupervised Learning

Data: $\langle \mathbf{x}_i \rangle$

Task: Learn **structure**

Example:



There are two types of apples

Reinforcement Learning

Data: $\langle \text{state}, \text{action} \rangle$

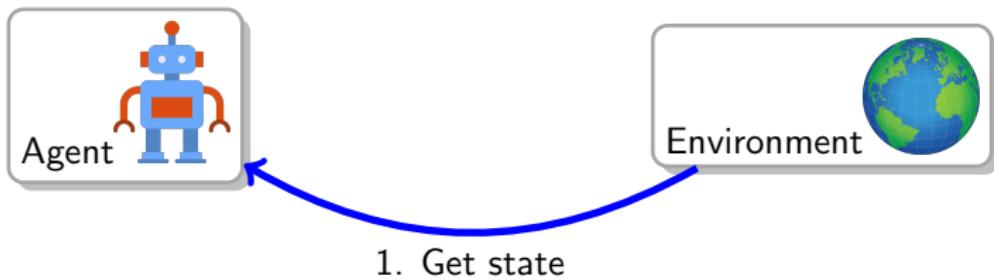
Task: Learn sequence of action to maximize reward

Example:

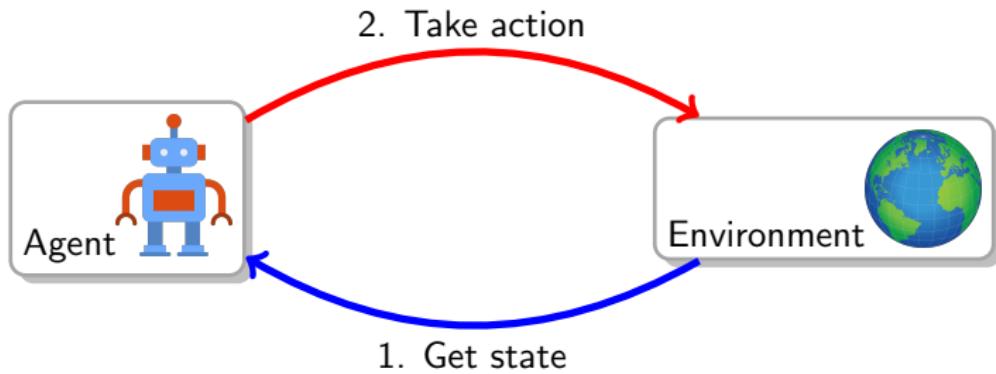


How to eat an apple

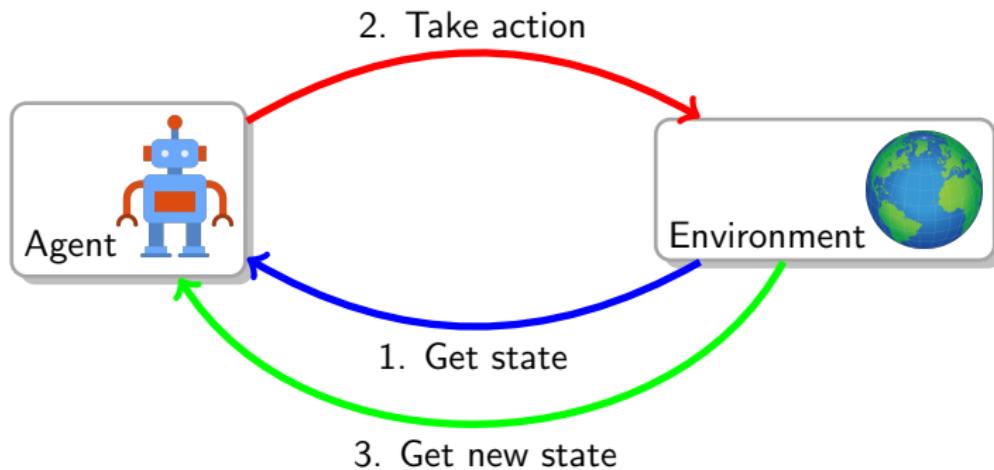
Agent in an environment



Agent in an environment



Agent in an environment



Examples

- ▶ Playing chess, Go, or many similar games
- ▶ Controller adjusting parameters of an engineered system (e.g., an oil refinery) in real time
- ▶ Robot learning to walk
- ▶ Everyday activities (e.g., making breakfast)

What is common?

- ▶ Handling the *whole* problem of a goal-oriented agent in an uncertain environment
- ▶ Trade-off between *exploitation* and *exploration*

Elements of a Reinforcement Learning Problem

- ▶ Agent operating in an environment
- ▶ State of the agent at time t - $S_t \in \mathcal{S}$
- ▶ Action taken by agent at time t - $A_t \in \mathcal{A}(S_t)$
- ▶ Reward at time t - $R_t \in \mathcal{R}$
- ▶ Policy - π (decision making rules)
 - ▶ $\pi(s) : \mathcal{S} \longrightarrow \mathcal{A}$
 - ▶ Action at a given state

Goal of RL

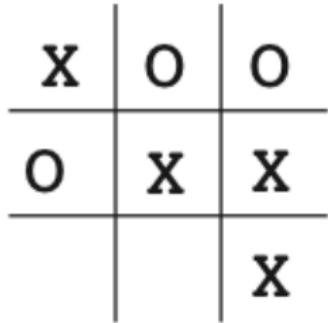
Learn optimal policy π that maximizes the reward

The Learning in Reinforcement Learning

1. Policy
2. Reward signal - used by the environment to inform the agent of the *reward* at a given time step
 - ▶ Primary basis for altering policy
 - ▶ Generally are functions of the current state and the actions
3. Value function - Expected total reward starting at a given state
 - ▶ Indicates the *long-term* desirability of a state
 - ▶ A state might have a small reward but a high value - might be followed by a sequence of states with high rewards
 - ▶ Harder to determine
4. Model - Allows us to model the environment (predict next states and next rewards)
 - ▶ Helps in planning

Learning to play Tic-Tac-Toe

- ▶ Other player is the environment
- ▶ **Task:** Construct a player that maximizes the chance of winning



Challenges

- ▶ Cannot assume a particular way of playing by the opponent
- ▶ Even then, you would need a completely specified model of the environment
- ▶ A possible approach – *learn* the model of the opponent's behavior by playing games against the opponent
- ▶ Or an exhaustive or evolutionary approach

Learning Tic-Tac-Toe the RL way

- ▶ State of the game is the configuration of 3×3 grid
- ▶ There can be 9^3 possible states
 - ▶ Of course many are trivial
- ▶ Value of each state is the probability of winning from that state
- ▶ Set the value of the *obvious* states as 1, others 0.5
 - ▶ e.g., assuming we move X, the value of the shown state is 1
- ▶ Play many games with the opponent
 - ▶ With probability $(1 - \delta)$ choose a move that results in the highest value state (*exploitation*)
 - ▶ With probability δ choose a random move (*exploration*)

X	O	O
O	X	X
		X

Updating value

- ▶ After each greedy move, use the value of current state ($V(S_{t+1})$) to update the value of the previous state:

X	O	O
O	X	X
		X

$$V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$$

- ▶ where α is a small positive fraction called the *step-size parameter*.
- ▶ An example of a *temporal-difference* learning method

Difference from evolutionary methods

Evolutionary

- ▶ Operates at a policy level
- ▶ Evaluates a policy over many games and then chooses the next policy
- ▶ For each game, only the final outcome is considered

Value function learning

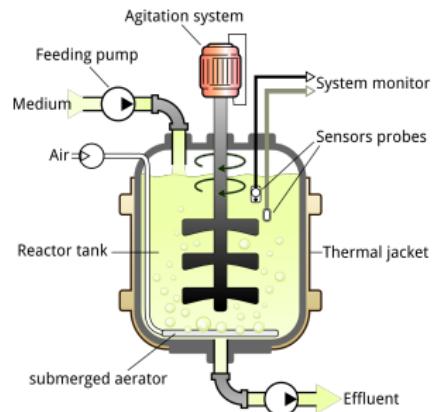
- ▶ Evaluates individual states during the course of play

Some Realistic Examples I

- ▶ **Setting:** Bioreactor producing useful chemicals
 - ▶ **Task:** Set optimal temperature and stirring rates during the operation

RL Setup

- ▶ **State:** Sensory readings (temperature, chemical composition)
 - ▶ **Action:** Change temperature and/or stirring rate
 - ▶ **Reward:** Measure of the useful chemical produced



Some Realistic Examples II

- ▶ **Setting:** Robot picking up an object
 - ▶ **Task:** Give optimal actuator inputs to the robot to enable smooth picking

RL Setup

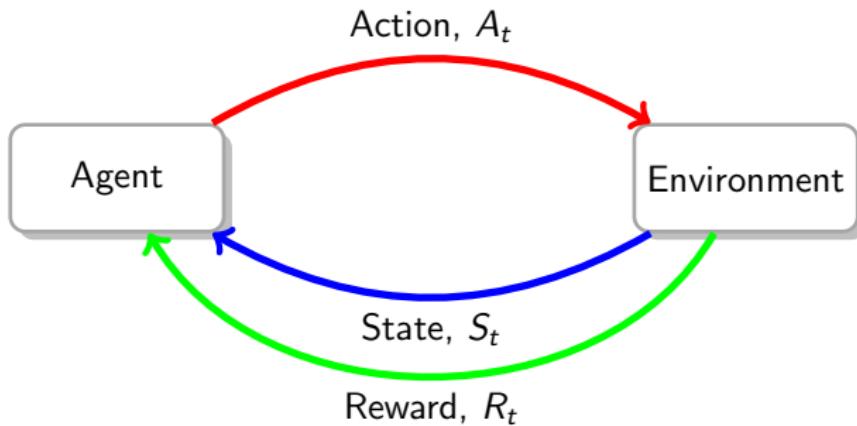
- ▶ **State:** Readings of joint angles and velocities
 - ▶ **Action:** Change voltages to motors at each joint
 - ▶ **Reward:** +1 if object picked up successfully
 - ▶ Additionally, one could give a small reward at each step if the motion is *not jerky*



Markov Decision Processes (MDP)

- ▶ A mathematically idealized formulation of the RL problem
 - ▶ Can be analyzed theoretically
- ▶ Allows one to probabilistically reason about next state and reward, given the current state and action
- ▶ A wider range of RL applications can be formulated as finite MDPs
- ▶ But there are other ways beyond MDP

Markov Decision Processes (MDP) I



Environment state	$S_t \in \mathcal{S}$
Agent action	$A_t \in \mathcal{A}$
Reward	$R_t \in \mathcal{R}$

Markov Decision Processes (MDP) II

- ▶ The agent-environment interaction gives rise to a sequence or *trajectory*:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots$$

- ▶ Finite MDP assumes that $\mathcal{S}, \mathcal{A}, \mathcal{R}$ are finite
- ▶ A joint probability distribution can be defined for (s', r) , where $s' \in \mathcal{S}$ and $r \in \mathcal{R}$:

$$p(s', r|s, a) \doteq P(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$

- ▶ p defines the *dynamics* of the MDP
 - ▶ A four argument function: $\mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

Why Markovian?

- ▶ The probabilities given by p completely characterizes the environment's dynamics
 - ▶ Probability of each possible value for S_t and R_t depends on the S_{t-1} and A_{t-1} , and nothing before

Versatility of dynamics function p

- ▶ Reveals “everything” about the environment:

1. State-transition probabilities:

$$p(s'|s, a) \doteq \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

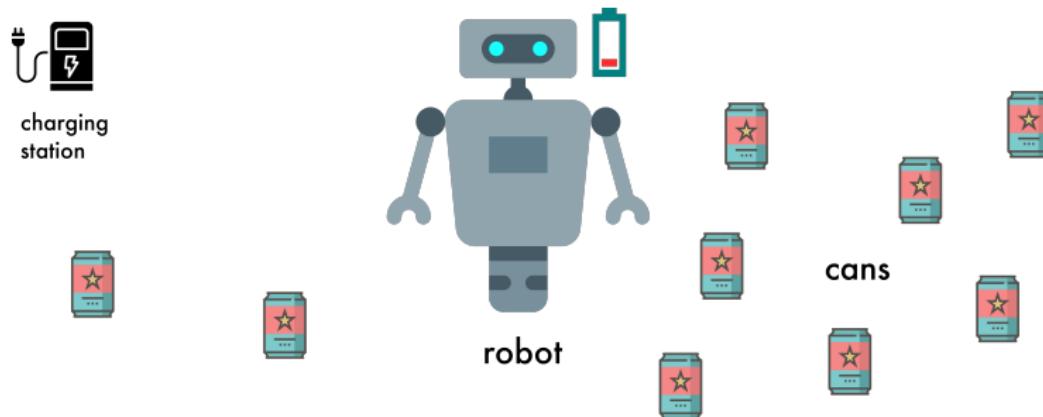
2. Expected reward for a given state-action pair:

$$r(s, a) \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$$

3. Expected rewards for state-action-next-state triplet:

$$r(s, a, s') \doteq \mathbb{E}[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s, a)}{p(s'|s, a)}$$

Example - A can picking robot I



- ▶ $S = \{\text{high}, \text{low}\}$ (battery level)
- ▶ $\mathcal{A} = \{\text{search}, \text{wait}, \text{recharge}\}$
 - ▶ $\mathcal{A}(\text{high}) = \{\text{search}, \text{wait}\}$
 - ▶ $\mathcal{A}(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

Example - A can picking robot II

State Transitions

- ▶ While searching, if state is high, the probability of the state to stay high is α and to become low is $1 - \alpha$.
- ▶ While searching, if the state is low, the probability of the state to stay low is β and battery to become depleted is $1 - \beta$
 - ▶ In this case, the robot is rescued and charged back to high

Rewards

- ▶ Positive rewards (+1) if the robot finds a can
- ▶ Negative (-3) if the battery runs down and the robot has to be rescued
- ▶ Expected reward when searching is r_{search} and when waiting is r_{wait}
- ▶ $r_{\text{search}} > r_{\text{wait}}$
- ▶ No cans collected when returning to recharge or when battery is depleted

Example - A can picking robot III

- ▶ The above system is a finite MDP

s	a	s'	$p(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

- ▶ Some transitions have zero probability of occurring, so no expected reward is specified for them

Where is the learning?

- ▶ In the above example, the MDP was *fully specified*
- ▶ We can plan the optimal behavior of the agent, given this data
- ▶ But usually we will not be this lucky
 - ▶ Need to learn some or all of the probabilities in MDP
 - ▶ Will need an objective function
 - ▶ A mathematical definition of the *cumulative reward*

Two types of tasks I

Episodic Tasks

- ▶ Agent-environment interaction can be broken into subsequences or *episodes*.
 - ▶ Plays of a game, trips through a maze, or any other repeated interaction
- ▶ Each episode ends in a terminal states
- ▶ Next episode begins independently of how the previous one ended
- ▶ \mathcal{S} is usually used to denote set of all *non-terminal* states
- ▶ \mathcal{S}^+ denotes the set of all states (including non-terminal)

Continuing Tasks

- ▶ Agent-environment interaction cannot be naturally broken into identifiable episodes
 - ▶ Any life-long learning task

Formal Definition of Learning Goal

- ▶ Maximize the *expected return*

Expected Return

$$G_t \doteq R_{t+1} + R_{t+2} + \dots + R_T$$

where T is the final time step.

- ▶ The above formulation does not work for *continuing tasks* where $T = \infty$

Expected Return with Discounting

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots + R_T$$

where γ is the *discount rate parameter*, $0 \leq \gamma \leq 1$.

Significance of the discount rate

- ▶ If $\gamma = 0$, the agent is only maximizing the immediate reward (short-sighted)
- ▶ As γ tends to 1, the agent gives more weight to future rewards
- ▶ Connection between successive expected returns:

$$G_t = R_{t+1} + \gamma G_{t+1}$$

- ▶ While the expected return is a sum of an infinite series, it will have a finite value if $\gamma < 1$
- ▶ Example, if $R_t = +1, \forall t$

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1 - \gamma}$$

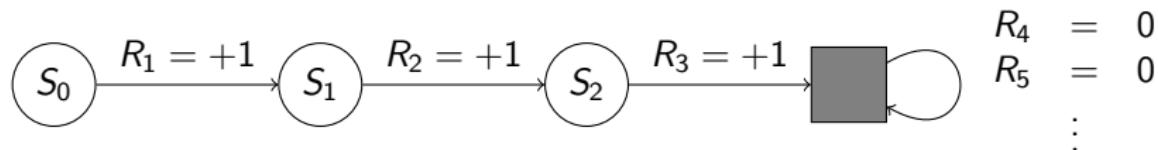
Unifying the two types of learning

- ▶ There are two types of learning tasks - episodic and continuing
- ▶ Episodic is mathematically easier, since each action only affects a finite number of subsequent rewards
- ▶ However, we will first come up with a unified notation for both

Episodic Tasks

- ▶ Technically, state representation at time t should be written as $S_{t,i}$, where i denotes the i^{th} episode
- ▶ However, we will drop the i subscript since we will not consider more than one episode at a time

Add an absorbing state



- ▶ We add a special *absorbing state* at the end of the episode
 - ▶ Transitions to itself
 - ▶ Generates reward of 0
- ▶ The reward sequence above will be $+1, +1, +1, 0, 0, 0, \dots$
- ▶ Allows for a unified expression for the expected return at t :

$$G_t \doteq \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

where T can be ∞ or $\gamma = 1$, but not both

What does learning in RL entail?

- ▶ Need to estimate the *value function*
 - ▶ At the current state, *how good* is a certain action (how good \equiv expected return)
- ▶ Need to have a policy

Policy

- ▶ A mapping from states to probabilities for selecting each possible action
- ▶ $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$
- ▶ This is what the agent uses to decide on the next action
- ▶ Given a policy, we can calculate the expected return, starting from state s
 - ▶ Also referred to as the *value function* at state s

Value Functions

State-value function

- ▶ Defined as the expected return when starting at state s and following the policy π :

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right], \forall s \in \mathcal{S}$$

- ▶ Also, referred to as the *state-value function* for policy, π

Action-value function

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]$$

Can one learn an optimal policy, given the complete model of the environment? I

- ▶ Not a very “useful” question to ask.
- ▶ But is important for general understanding
- ▶ We assume that the dynamics of the environment (model) are known, i.e., we know $p(s', r|s, a), \forall s', s \in \mathcal{S}, a \in \mathcal{A}(s), r \in \mathcal{R}$
- ▶ There is always an optimal policy, π_* that is better than all other policies - gives the optimal returns for all states
- ▶ The optimal policy will have corresponding optimal value functions, $v_*(s)$ and $q_*(s, a)$:

$$v_*(s) \doteq \max_{\pi} v_{\pi}(s), \forall s \in \mathcal{S}$$

and

$$q_*(s, a) \doteq \max_{\pi} q_{\pi}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

- ▶ We can identify the optimal policy, given the optimal value functions, $v_*(s)$ and $q_*(s)$

Can one learn an optimal policy, given the complete model of the environment? II

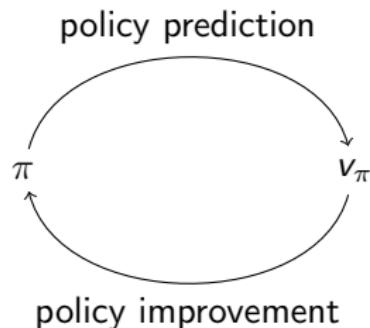
Policy Prediction

$$\begin{aligned}v_{\pi}(s) &\doteq \mathbb{E}_{\pi}[G_t | S_t = s] \\&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \\&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma \mathbb{E}[G_{t+1} | S_{t+1} = s']] \\&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) [r + \gamma v_{\pi}(s')], \forall s \in \mathcal{S}\end{aligned}$$

- ▶ A set of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ unknowns

Can one learn an optimal policy, given the complete model of the environment? III

- ▶ One can use a *Dynamic Programming* (DP) formulation to find the optimal value functions and the optimal policy
- ▶ Not further discussed in this class - See Chapter 4 of Sutton and Barto



Can one learn an optimal policy, given an incomplete model of the environment? I

- ▶ Monte Carlo Methods
- ▶ Focusing on *episodic learning*

Step 1 - Policy Prediction

- ▶ We want to estimate $v_\pi(s)$, i.e., the value of a state under a given policy π
- ▶ “Play the game” N times
- ▶ Maintain an average of the returns observed after observing a state (denoted by $\hat{v}_\pi(s)$)
 - ▶ Two options: *first-visit* and *every-visit*
- ▶ As $N \rightarrow \infty$, $\hat{v}_\pi(s) \rightarrow v_\pi(s)$
- ▶ Can learn q_π in the same way
- ▶ Not very efficient if doing this for every state



Can one learn an optimal policy, given an incomplete model of the environment? II

First-visit MC prediction

Input: a policy π to be evaluated

Initialize:

- ▶ $V(s) \in \mathbb{R}$, arbitrarily, $\forall s \in \mathcal{S}$
- ▶ $Returns(s) \leftarrow []$, $\forall s \in \mathcal{S}$

Loop forever (for each episode):

- ▶ Generate episode using π : $S_0, A_0, R_1, S_1, A_1, \dots, S_{T-1}, A_{T-1}, R_T$
- ▶ $G \leftarrow 0$
- ▶ Loop for each step, $t = T - 1, T - 2, \dots, 0$:
 - ▶ $G \leftarrow \gamma G + R_{t+1}$
 - ▶ If $S_t \notin S_0, S_1, \dots, S_{t-1}$:
 - ▶ $\text{append}(Returns(S_t), G)$
 - ▶ $V(S_t) \leftarrow \text{average}(Returns(S_t))$

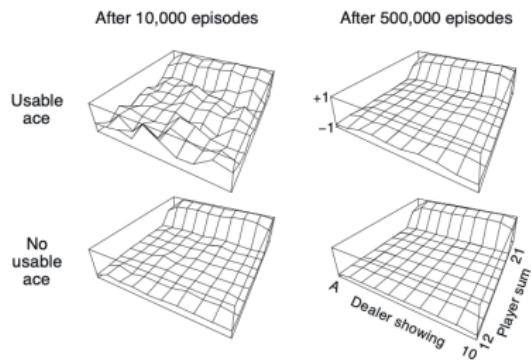
Playing Blackjack I



- ▶ Objective is to get cards with numerical values as close to 21 without exceeding it
- ▶ Playing against the house/dealer (environment)
- ▶ Can be formulated as an episodic MDP
 - ▶ One game is an episode
- ▶ Actions: hit or stick
- ▶ Rewards: 0 (*draw* or during the game), +1 (player wins), -1 (player loses)

Playing Blackjack II

- ▶ States: 200 possible states, which are combinations of
 1. Current sum of player cards (between 12-21)
 2. Dealer's face-up card (between 1 and 11)
 3. Is ace *usable* (two possible values)
- ▶ Consider the following policy for the player (π):
 - ▶ Player sticks if the sum of cards is 20 or 21
 - ▶ Otherwise hits
- ▶ What is the state-value function for this policy?



Complete Monte Carlo Method

- ▶ One can also estimate *action values* or values of state-action pairs, or $q_\pi(s, a)$
 - ▶ When model is not available, i.e., we do not have $p(s', r|s, a)$, $v_\pi(s)$ is not enough to determine the policy (as is done with *dynamic programming* or DP)
 - ▶ $q_\pi(s, a)$ is needed
- ▶ Then use an iterative scheme to update the “optimal” policy based on the value function
 - ▶ Also known as *policy control*

And finally, the Temporal-Difference (*TD*) Learning I

- ▶ Does not need the model of the Environment
- ▶ Differs from DP and Monte-carlo method in the *policy prediction* step.
- ▶ The control problem is solved using an iterative scheme, similar to DP and Monte-carlo method

TD Prediction

- ▶ Already seen a variant in the beginning of this topic

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

- ▶ Update $V(S_t)$ immediately after transition to S_{t+1} and receiving R_{t+1}
- ▶ This is known as *TD(0)* or *one-step TD*

And finally, the Temporal-Difference (TD) Learning II

Tabular $TD(0)$ for estimating v_π

Input: a policy π to be evaluated

Parameter: step size $\alpha, \gamma \in (0, 1]$ *Initialize:*

$V(s) \in \mathbb{R}$, arbitrarily, $\forall s \in \mathcal{S}$, except that $V(\text{terminal}) = 0$ *Loop for each episode:*

- ▶ Initialize S
- ▶ *Loop for each step of episode:*
 - ▶ $A \leftarrow$ action given by π for S
 - ▶ Take action A , observe R and S'
 - ▶ $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
 - ▶ $S \leftarrow S'$
- ▶ Until S is terminal

TD Policy Control

- ▶ Also iterative, as DP and Monte-Carlo
- ▶ Two variants:
 - ▶ *On-policy control* (SARSA)
 - ▶ Estimate q_π for current policy π using the current action according to the policy
 - ▶ *Off-policy control* (Q-learning)
 - ▶ Estimate q_π for current policy π using the current optimal action at the given state

To conclude

- ▶ Reinforcement learning (RL) allows us to solve learning problems that cannot be solved using traditional supervised and unsupervised ML
- ▶ Markov Decision Processes (MDP) are one way to formally represent an RL problem
- ▶ We have seen three different ways to do learning in the context of MDP
 1. Dynamic programming (DP)
 2. Monte-carlo methods
 3. Temporal differencing (TD)
- ▶ These are all examples of *tabular methods*
- ▶ Monte-carlo and TD are data driven and do not assume knowledge of the model for the environment

References