CSC 411 Lectures 21–22: Reinforcement Learning

Roger Grosse, Amir-massoud Farahmand, and Juan Carrasquilla

University of Toronto

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.
- In many cases, we want to take a sequence of actions, each of which affects the future possibilities, i.e., the actions have long-term consequences.
- We want to solve sequential decision-making problems using learning-based approaches.

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_l4U9A

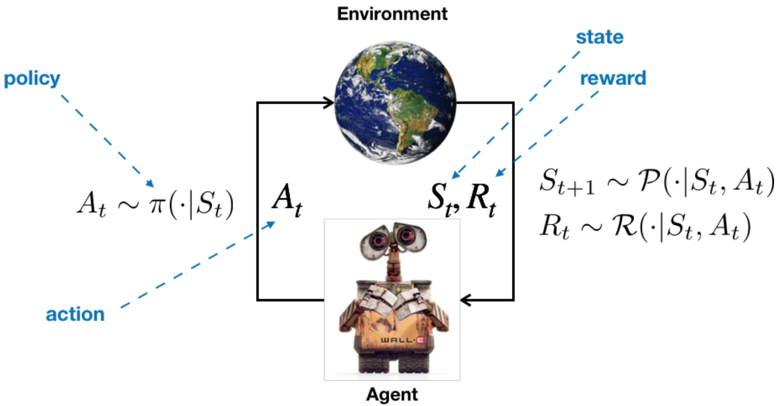
Making Pancakes!

https://www.youtube.com/watch?v=W_gxLKSsSIE

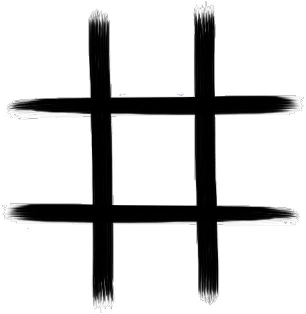
Reinforcement Learning

- Learning problems differ in the information available to the learner:
  - Supervised: For a given input, we know its corresponding output, e.g., class label
  - Reinforcement learning: We observe inputs, and we have to choose outputs (actions) in order to maximize rewards. Correct outputs are not provided.
  - Unsupervised: We only have input data. We somehow need to organize them in a meaningful way, e.g., clustering.
- 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

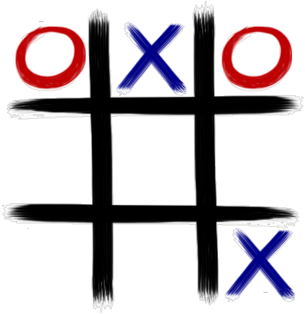


Example: Tic Tac Toe, Notation



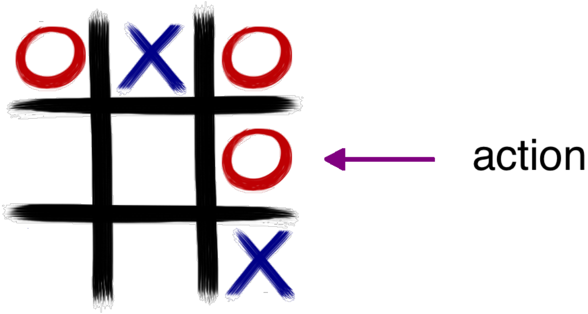
environment

Example: Tic Tac Toe, Notation

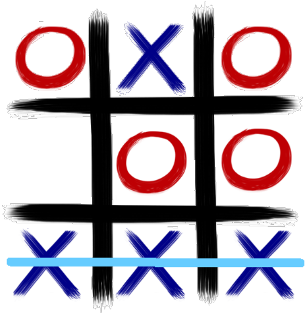


(current)
state

Example: Tic Tac Toe, Notation



Example: Tic Tac Toe, Notation



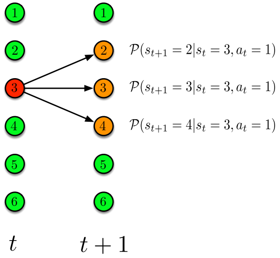
reward
(here: -1)

Formalizing Reinforcement Learning Problems

- 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., $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$.
  - $\mathcal{P}$: Transition probability
  - $\mathcal{R}$: Immediate reward distribution
  - γ: Discount factor $(0 \leq \gamma < 1)$
- Let us take a closer look at each of them.

Formalizing Reinforcement Learning Problems

- The agent has a state $s \in \mathcal{S}$ in the environment, e.g., the location of X and O in tic-tac-toc, 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 $R_{t+1} \sim \mathcal{R}(\cdot|S_t, A_t, S_{t+1})$



Formulating Reinforcement Learning

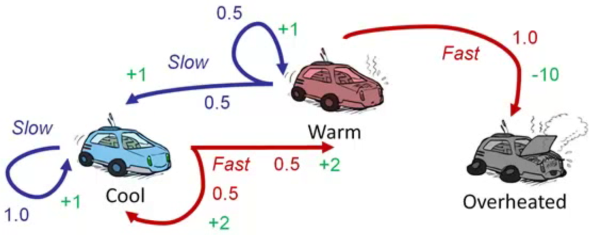
- The action selection mechanism is described by a policy π
  - Policy π is a mapping from states to actions, i.e., $A_t = \pi(S_t)$ (deterministic) or $A_t \sim \pi(\cdot|S_t)$ (stochastic).
- The goal is to find a policy π such that long-term rewards of the agent is maximized.
- Different notions of the long-term reward:
  - Cumulative/total reward: $R_0 + R_1 + R_2 + \dots$
  - Discounted (cumulative) reward: $R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots$
    - The discount factor $0 \leq \gamma \leq 1$ determines how myopic or farsighted the agent is.
    - When γ is closer to 0, the agent prefers to obtain reward as soon as possible.
    - When γ is close to 1, the agent is willing to receive rewards in the farther future.
    - The discount factor γ has a financial interpretation: If a dollar next year is worth almost the same as a dollar today, γ is close to 1. If a dollar's worth next year is much less its worth today, γ 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^π (or simply value function) for policy π is a function defined as

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

  It describes the expected discounted reward if the agent starts from state s and follows policy π.
- The action-value function Q^π for policy π 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 π.

Value Function

- The goal is to find a policy π 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

$$\pi^*(s) \leftarrow \operatorname*{argmax}_{a} Q^*(s, a)$$

- The goal of an RL agent is to find a policy π 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
- Let us take a closer look at the value function

Bellman Equation

The value function satisfies the following recursive relationship:

$$
\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\right] \\
&= \mathbb{E}\left[R(S_0, A_0) + \gamma \sum_{t=0}^{\infty} \gamma^t R_{t+1} | s_0 = s, a_0 = a\right] \\
&= \mathbb{E}\left[R(S_0, A_0) + \gamma Q^\pi(S_1, \pi(S_1)) | S_0 = s, A_0 = a\right] \\
&= \underbrace{r(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s'|s,a) Q^\pi(s', \pi(s'))}_{\triangleq (T^\pi Q^\pi)(s,a)}
\end{aligned}
$$

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

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

Bellman Equation

- Key observation:

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

- 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:

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

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

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

Finding the Value Function

- Let us first study the policy evaluation problem: Given a policy π, find V^π (or Q^π).
- 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 $r(s,a) = \mathbb{E}\left[\mathcal{R}(\cdot|s,a)\right]$ are known.
- 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:

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

- 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 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:

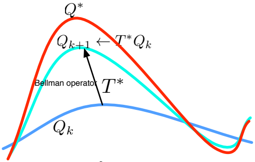
$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \max_{a' \in \mathcal{A}} Q(s', a') \qquad \forall (s, a) \in \mathcal{S} \times \mathcal{A}$$

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

Finding the Optimal Value Function: 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 $k = 1, 2, \ldots,$ apply

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

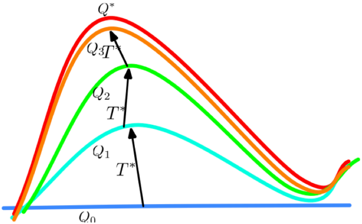


$$Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s' | s, a) \max_{a' \in \mathcal{A}} Q_k(s', a')$$

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

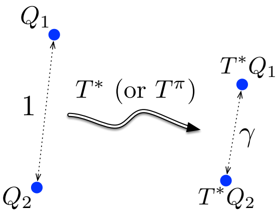
Value Iteration

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



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

Bellman Operator is Contraction (Optional)



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

Bellman Operator is Contraction (Optional)

Therefore, we get that

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

Or more succinctly,

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

We also have a similar result for the Bellman operator of a policy π:

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

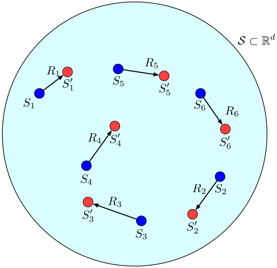
Challenges

- 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 $Q_{k+1}(s, a) \leftarrow r(s, a) + \gamma \int_{\mathcal{S} \times \mathcal{A}} \mathcal{P}(\mathrm{d}s'|s, a) \max_{a' \in \mathcal{A}} Q_k(s', a')$
- We often do not know the dynamics $\mathcal{P}$ and the reward function $\mathcal{R}$, so we cannot calculate the Bellman operators.

Is There Any Hope?

- During this course, we learned 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 technique 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

$$
\begin{aligned}
&\mathcal{D}_n = \{(S_i, A_i, R_i, S_i')\}_{i=1}^n \\
&(S_i, A_i) \sim \nu \qquad (\nu \text{ is a distribution over } \mathcal{S} \times \mathcal{A}) \\
&S_i' \sim \mathcal{P}(\cdot|S_i, A_i) \\
&R_i \sim \mathcal{R}(\cdot|S_i, A_i)
\end{aligned}
$$

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

From Value Iteration to Approximate Value Iteration

- Recall that each iteration of VI computes

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

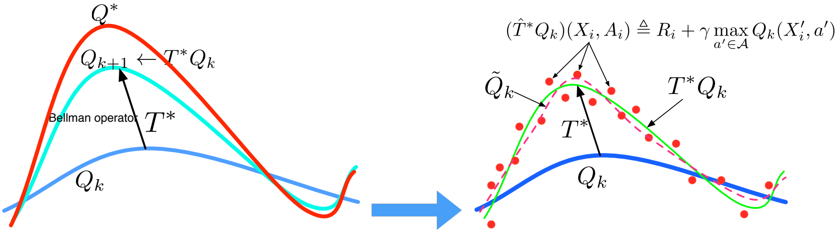
- We cannot directly compute $T^* Q_k$. But we can use data to approximately perform one step of VI.
- Consider (S_i, A_i, R_i, S'_i) from the dataset $\mathcal{D}_n$.
- Consider a function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
- We can define a random variable $t_i = R_i + \gamma \max_{a'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

$$\mathbb{E}\left[R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a') | S_i, A_i \right] =$$

$$r(S_i, A_i) + \gamma \int \mathcal{P}(\mathrm{d}s' | S_i, A_i) \max_{a' \in \mathcal{A}} Q(s', a') = (T^* Q)(S_i, A_i)$$

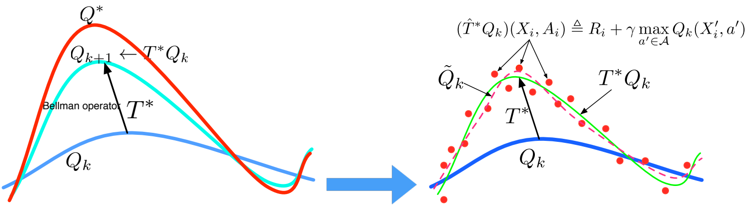
- So $t_i = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S'_i, a')$ is a noisy version of $(T^* Q)(S_i, A_i)$. Fitting a function to noisy real-valued data is the regression problem.

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 $\{(\mathbf{x}^{(i)}, t^{(i)})\}_{i=1}^N$ with $\mathbf{x}^{(i)} = (S_i, A_i)$ and $t^{(i)} = R_i + \gamma \max_{a' \in \mathcal{A}} Q(S_i', a')$.
- Because of $\mathbb{E}\left[R_i + \gamma \max_{a' \in \mathcal{A}} Q_k(S_i', a') | S_i, A_i\right] = (T^* Q_k)(S_i, A_i)$ we can treat the problem of estimating Q_{k+1} as a regression problem with noisy data.

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 solve a regression problem. We minimize the squared error:

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

- 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: $\mathcal{F} = \{Q(s, a) = \mathbf{w}^\top \psi(s, a)\}$.
  - How to choose the feature mapping ψ?
- 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 only have $Q_{k+1} \approx T^* Q_k$.
- These errors might accumulate and may even cause divergence.
- The theoretical analysis of AVI is more complicated than the analysis of regression problems. But it has been done.

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 π_t be?
- Q-Learning is an online algorithm that addresses the first question.
- We present Q-Learning for finite state-action problems.

Q-Learning with ε-Greedy Policy

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

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

  - 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 ε-greedy is a simple mechanism for maintaining exploration-exploitation tradeoff.

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

- The ε-greedy policy ensures that most of the time (probability $1-\varepsilon$) 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 ε) it explores other actions.
- Without exploration, the agent may never find some good actions.
- The ε-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.

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]

An Intuition on Why Q-Learning Works? (Optional)

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

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

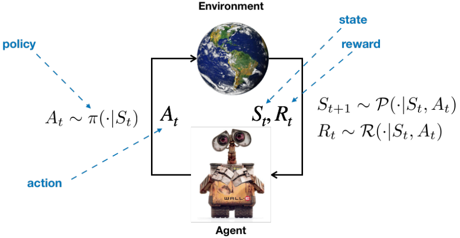
- To understand this better, let us focus on its stochastic equilibrium, i.e., where the expected change in $Q(S, A)$ is zero. We have

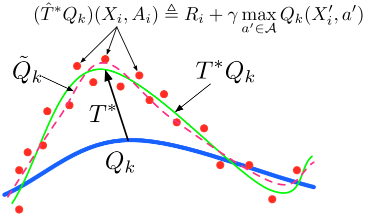
$$\begin{aligned} &\mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') - Q(S,A) | S, A \right] = 0 \\ \Rightarrow &(T^*Q)(S,A) = Q(S,A) \end{aligned}$$

- 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^*.
- This is true for finite state-action spaces. The equivalent of the Q-Learning with function approximation might diverge.

Recap and Other Approaches

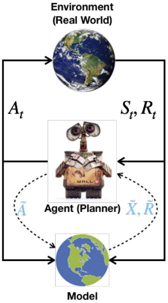
- We defined MDP as the mathematical framework to study RL problems.
- We started from the assumption that the model is known (Planning). We then relaxed it to the assumption that we have a batch of data (Batch RL). Finally we briefly discussed Q-learning as an online algorithm to solve RL problems (Online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

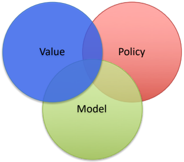




Recap and Other Approaches

- 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 estimate $\mathcal{P}$ 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 School, 2018.
  - Deep Reinforcement Learning, CS 294-112 at UC Berkeley