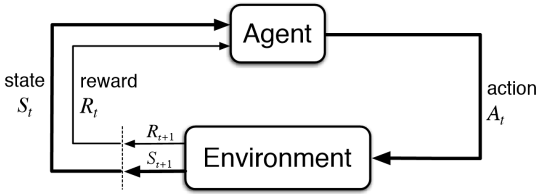
Reinforcement Learning

Dit-Yan Yeung

Department of Computer Science and Engineering
Hong Kong University of Science and Technology

COMP 4211: Machine Learning (Fall 2022)

Sequential Decision Problems



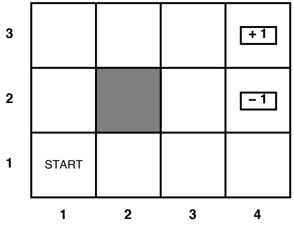
- Like the recurrent neural networks that we have studied, solving sequential decision problems involves making multiple decisions.
- However, a crucial difference is that a decision (also called action) made by the decision maker (also called agent) in a sequential decision problem can affect the environment and hence the state (i.e., future input) of the agent.

Optimal Decision Making

- The agent receives a reward (or called penalty for a negative reward) for the action it takes in a state.
- The goal is to maximize the total reward (also called cumulative reward) over a sequence of actions.
- A policy is a mapping from the set of states to the set of actions.
- The optimal policy gives a sequence of actions that maximize the total reward.
- Reinforcement learning is the learning paradigm (different from supervised learning and unsupervised learning we have studied) that solves sequential decision problems by learning to approximate the optimal policy.
- Examples of decision-making agents:
    - Chess or Go player
    - Mobile robot

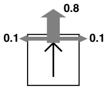
A Toy Problem

- A simple grid environment:



- The two terminal states, $(4, 3)$ and $(4, 2)$, have rewards $+1$ and -1, respectively, and all other states have a reward of -0.04.
- Starting from $(1, 1)$, a shorter path to $(4, 3)$ is preferred because visiting each nonterminal state induces a negative reward.

Stochastic State Transition



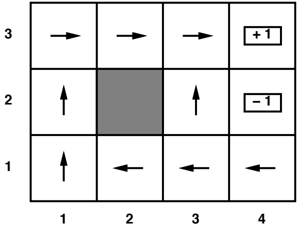
- The transition model is stochastic in the sense that the intended outcome of each action generally occurs with a probability < 1.
- The intended outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. Bumping into a wall results in no movement.
- Markovian transition model:

$$p(s' \mid s, a) = \text{probability that taking action } a \text{ in state } s \text{ leads to state } s'$$

- A sequential decision problem with a Markovian transition model and additive rewards is called a Markov decision process (MDP).

Optimal Policy

- Optimal policy:



- The optimal policy for $(3, 1)$ is conservative because the cost of taking a step is fairly small compared with the penalty for ending up in $(4, 2)$ by accident due to uncertainty of state transition.
- In general the optimal policy may change if the reward of the nonterminal states is not -0.04.

Markov Decision Processes

- An MDP describes an environment for reinforcement learning in which all states are Markovian and observable.
- If the states are not fully observable, it is called a partially observable Markov decision process (POMDP) which is more realistic but beyond the scope here.

- Formally, a (discrete) MDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ where:
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ specifies the state transition probabilities:

$$\mathcal{P}_{ss'}^{a} = \Pr\left\{S_{t+1} = s' \mid S_t = s, A_t = a\right\}.$$

  - $\mathcal{R}$ is a reward function:

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

  - $\gamma \in [0, 1]$ is a discount factor

Markov Property

- A state S_t satisfies the Markov property if and only if

$$\Pr\{S_{t+1} \mid S_t\} = \Pr\{S_{t+1} \mid S_1, \ldots, S_t\}.$$

- The state is a sufficient statistic of the future:
  "The future is independent of the past given the present".

Return

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

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

- It values immediate reward above delayed reward:
  - $\gamma \to 0$: "myopic" evaluation
  - $\gamma \to 1$: "far-sighted" evaluation
- $\gamma < 1$ ensures that the return is finite.
- $\gamma = 1$ may be used if the MDP is episodic, i.e., it always terminates.

Policy

- A (stochastic) policy π is a distribution over actions given a state:

$$\pi(a \mid s) = \Pr\{A_t = a \mid S_t = s\}.$$

- A policy fully defines the behavior of an agent.
- MDP policies are stationary, i.e., time-independent:

$$A_t \sim \pi(\cdot \mid S_t), \quad \forall t > 0.$$

- A deterministic policy is a policy which deterministically selects the action to take at the current state.

Value Functions

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

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

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

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

Bellman Expectation Equation

- The state-value function can be decomposed into two parts, the immediate reward R_{t+1} and the discounted value of successor state $\gamma v_\pi(s')$:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots) \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\Big] \\
&= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right] \\
&= \mathbb{E}_\pi\left[r + \gamma v_\pi(s') \mid S_t = s\right].
\end{aligned}
$$

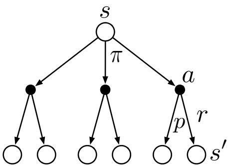
Bellman Expectation Equation (2)

- This recursive form is called the Bellman expectation equation.
- It can be derived similarly for the action-value function:

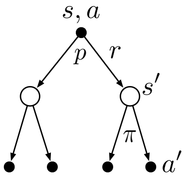
$$q_\pi(s, a) = \mathbb{E}_\pi\left[r + \gamma q_\pi(s', a') \mid S_t = s, A_t = a\right].$$

Backup Diagrams for v_π and q_π

- Backup diagram for v_π:



- Backup diagram for q_π:



Bellman Expectation Equation in Matrix Form

- The Bellman expectation equation can be expressed concisely in matrix form:

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

where v_π is a column vector with one entry per state, i.e.

$$\begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1^\pi \\ \vdots \\ \mathcal{R}_n^\pi \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11}^\pi & \cdots & \mathcal{P}_{1n}^\pi \\ \vdots & & \vdots \\ \mathcal{P}_{n1}^\pi & \cdots & \mathcal{P}_{nn}^\pi \end{bmatrix} \begin{bmatrix} v_\pi(1) \\ \vdots \\ v_\pi(n) \end{bmatrix}.$$

- It is a linear equation which can be solved directly as follows:

$$\begin{aligned} v_\pi &= \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi \\ (\mathbf{I} - \gamma \mathcal{P}^\pi) v_\pi &= \mathcal{R}^\pi \\ v_\pi &= (\mathbf{I} - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi. \end{aligned}$$

Optimal Value Functions

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

$$v_*(s) = \max_\pi v_\pi(s).$$

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

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

- The optimal state-value or action-value function specifies the best possible performance in a given MDP.

Optimal Policy

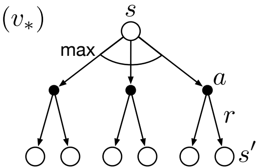
- We can define a partial ordering over all policies: $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \ \forall s$.
- For any MDP:
  - There always exists an optimal policy π_*:
$$\pi_* \geq \pi, \quad \forall \pi.$$
  - All optimal policies achieve the optimal state-value function:
$$v_{\pi_*}(s) = v_*(s).$$
  - All optimal policies achieve the optimal action-value function:
$$q_{\pi_*}(s, a) = q_*(s, a).$$
- An optimal policy can be found by maximizing over $q_*(s, a)$:
$$\pi_*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{otherwise.} \end{cases}$$

Bellman Optimality Equation and Backup Diagram for v_*

- Bellman optimality equation for v_*:

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

- Backup diagram for v_*:



- Cf. Bellman expectation equation for v_* which can be expressed as:

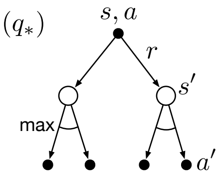
$$v_\pi(s) = \sum_a \pi(a \mid s) \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v_*(s') \right\}.$$

Bellman Optimality Equation and Backup Diagram for q_*

- Bellman optimality equation for q_*:

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

- Backup diagram for q_*:



- Since the Bellman optimality equations are nonlinear with no closed-form solutions in general, iterative algorithms are needed (to be discussed below).

Optimal State-Value Function for Toy Problem

- Values of states based on optimal policy with $\gamma = 1$:

3	0.812	0.868	0.912	$\boxed{+1}$
2	0.762		0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

- In general the values are higher for states closer to $(4, 3)$ because fewer steps are required to reach it.

Dynamic Programming

- Classical dynamic programming (DP) algorithms compute optimal policies, assuming full knowledge of the underlying MDP and the existence of sufficient computational resources.
- Note that this setting is of limited utility in most realistic reinforcement learning problems, but it is important theoretically and provides useful insights for more practical reinforcement learning algorithms.
- The key idea of DP, and of reinforcement learning in general, is the use of value functions to organize and structure the search for good policies.
- The optimal policies can be obtained after finding the optimal value functions v_* or q_*:

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

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

Update Rules Derived from Bellman Equations

- DP algorithms make use of update rules obtained by turning the equality ($=$) in the Bellman optimality equations into assignment ($\leftarrow$), which aims to iteratively improve approximations of the desired value functions.
- For example, we want to construct a sequence $\{v_k\}$ that converges asymptotically to v_* based on the following update rule:

$$v_{k+1}(s) \leftarrow \max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v_k(s') \right\}.$$

- This update rule is used in the value iteration algorithm.

Value Iteration Algorithm for Estimating $\pi \approx \pi_*$ Based on v

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

$v(s) \leftarrow 0$ for all states s

repeat

  $\delta \leftarrow 0$

  for each state s **do**

    $v_{prev} \leftarrow v(s)$

    $v(s) \leftarrow \max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v(s') \right\}$

    $\delta \leftarrow \max(\delta, |v_{prev} - v(s)|)$

  end for

until $\delta < \theta$

Output a deterministic policy $\pi \approx \pi_*$ s.t. $\pi(s) = \arg\max_a \left\{ \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \, v(s') \right\}$

Value Iteration Algorithm for Estimating $\pi \approx \pi_*$ Based on q

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
$q(s, a) \leftarrow 0$ for all state-action pairs (s, a)
repeat
  $\delta \leftarrow 0$
  for each state-action pair (s, a) **do**
    $q_{prev} \leftarrow q(s, a)$
    $q(s, a) \leftarrow \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q(s', a')$
    $\delta \leftarrow \max(\delta, |q_{prev} - q(s, a)|)$
  end for
until $\delta < \theta$
Output a deterministic policy $\pi \approx \pi_*$ s.t. $\pi(s) = \arg\max_a q(s, a)$

- Note that determining the optimal policy from q is easier than v.

Some Computational Issues

- Let $n = |\mathcal{S}|$ and $m = |\mathcal{A}|$ denote the number of states and actions, respectively, in an MDP.
- The total number of deterministic policies is m^n, which is exponential in n.
- However, DP algorithms such as value iteration can find an optimal policy in time polynomial in n and m.
- Despite the polynomial complexity, DP algorithms are still not practical for very large problems found in practical applications.

Model-Free Reinforcement Learning

- Unlike DP algorithms for solving MDPs, model-free reinforcement learning algorithms assume no knowledge of $\mathcal{P}$ and $\mathcal{R}$.
- For some problems, the MDP is actually known but is too big to apply DP algorithms.
- Model-based reinforcement learning algorithms aim to learn directly from episodes of experience interacting with the environment, requiring sufficient exploration in addition to exploitation.
- We will consider both tabular methods for discrete states and actions and function approximation methods for the continuous extension.

Q-Learning Algorithm

$q(s, a) \leftarrow 0$ for all state-action pairs (s, a)
repeat
  Initialize s
  repeat
    Choose action a at state s according to some strategy
    Take action a at state s, then observe s' and r
    $q(s, a) \leftarrow q(s, a) + \alpha\big[r + \gamma \max_{a'} q(s', a') - q(s, a)\big]$
    $s \leftarrow s'$
  until s is a terminal state
until convergence

Q-Learning Target and Temporal-Difference Learning

- Q-learning target based on a one-step look-ahead:

$$r + \gamma \max_{a'} q(s', a').$$

- Update rule based on temporal-difference (TD) learning:

$$q(s,a) \leftarrow q(s,a) + \alpha \Big[r + \gamma \max_{a'} q(s', a') - q(s,a) \Big],$$

  where α is the learning rate.

- It has been shown that the Q-learning update rule converges to the optimal action-value function, i.e., $q(s,a) \to q_*(s,a)$.

ϵ-Greedy Exploration

- To choose an action at a state, one possibility is the greedy algorithm, i.e., choose the action $\arg\max_a q(s, a)$.
- However, the greedy algorithm, which corresponds to a deterministic policy, may not provide sufficient exploration to improve the policy.
- ϵ-greedy search:

$$\pi(a \mid s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a = \arg\max_a q(s, a) \\ \epsilon/m & \text{otherwise.} \end{cases}$$

  - Explore: with probability ϵ, it chooses an action uniformly at random from all m possible actions.
  - Exploit: with probability $1 - \epsilon$, it chooses the best action so far.

Value Function Approximation

- So far we have represented the value function by a lookup table:
  - Every state s has an entry $v(s)$.
  - Or, every state-action pair (s, a) has an entry $q(s, a)$.
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory.
  - It is too slow to learn the value of each state individually.
- Solution for large MDPs:
  - Estimate the value function with function approximation:

$$\hat{v}(s; \mathbf{w}) \approx v_\pi(s)$$
$$\text{or} \quad \hat{q}(s, a; \mathbf{w}) \approx q_\pi(s, a).$$

  - Update the parameter $\mathbf{w}$ using a learning algorithm such as Q-learning.
  - Generalize from seen states to unseen states.
- Differentiable function approximators are preferred, e.g., linear combinations of features, neural networks.

Deep Q-Network

- The deep Q-network (DQN) is a combination of Q-learning with a deep convolutional neural network (CNN).
- Specifically, the action-value function is implemented using a deep CNN $q(s, a; \mathbf{w})$, with network weights represented as the vector $\mathbf{w}$.
- The update rule of Q-learning is changed to the following semi-gradient form:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \left[r + \gamma \max_{a'} q(s', a'; \mathbf{w}) - q(s, a; \mathbf{w}) \right] \nabla_{\mathbf{w}} q(s, a; \mathbf{w}).$$

To Learn More...

- Policy gradient algorithms