Reinforcement Learning

Ziping Zhao

School of Information Science and Technology
ShanghaiTech University, Shanghai, China

CS182: Introduction to Machine Learning (Fall 2021)
http://cs182.sist.shanghaitech.edu.cn

Outline

Outline

Introduction

Elements of Reinforcement Learning

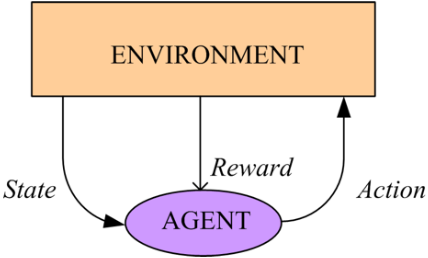
Model-Based Learning

Model-Free Learning

Deep Reinforcement Learning

Sequential Decision Problems

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

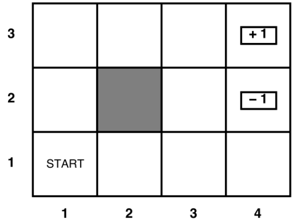


Optimal Decision Making

- The agent receives a reward (or penalty for a negative reward) for the action it takes in a state.
- The goal is to maximize the total reward (or 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 (RL) is the learning paradigm (different from supervised learning and unsupervised learning) that solves sequential decision problems by learning to approximate the optimal policy.
- Examples of decision-making agents:
  - Chess or Go player
  - Mobile robot
  - Electronic game player
  - Financial investor
  - etc.

A Toy Problem

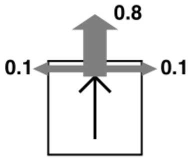
- A simple 4×3 grid environment:



- The two terminal states, (4, 3) and (4, 2), have reward $+1$ 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.
- If the environment were deterministic, a solution would be easy: [Up, Up, Right, Right, Right].

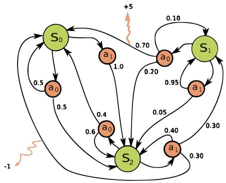
Stochastic State Transition

- The environment is stochastic/nondeterministic, i.e., 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:

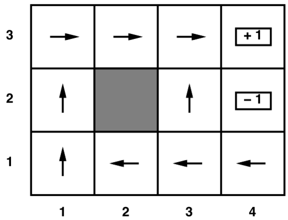
$$P(s_j \mid s_i, a) = \text{probability of taking action } a \text{ (like moving upward) in state } i \text{ leads to state } j$$



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

Optimal Policy

- Optimal policy (for infinite horizon) is given in the right.
- 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.

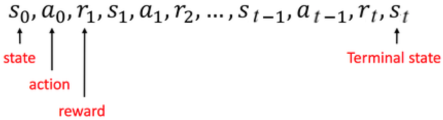


- In general the optimal policy may change if the reward of the nonterminal states is not -0.04.
- Finite horizon: fixed time T after which nothing matters (think of this as a deadline)
  - Suppose our agent starts at (3,1) and $T = 3$. Then to get to the $+1$ state, agent must go up.
  - If $T = 100$, agent can take the safe route around.
- For a finite horizon, the optimal action in a state can change over time.

Outline

Elements of Reinforcement Learning

- s_t: state of environment at time t; $\mathcal{S}$: set of all possible states
- a_t: action taken by agent at time t; $\mathcal{A}(s_t)$: set of possible actions in state s_t
- r_{t+1}: reward received after taking action a_t in state s_t, followed by transition to the next state s_{t+1}



- $P(s_{t+1} \mid s_t, a_t)$: state transition probability
- $p(r_{t+1} \mid s_t, a_t)$: reward probability
- Markov process: the state and reward in the next time step depend only on the current state and action.
- In some applications, reward and next state are deterministic, i.e., for a certain state and action taken, there is one possible reward value and next state.
- Episode or trial: the sequence of actions from the initial state to the absorbing terminal (goal) state.

Policy and Cumulative Reward

- Policy function (for deterministic policy):

$$\pi : \mathcal{S} \to \mathcal{A}$$

  which defines the behavior of an agent as a mapping from the states to the actions.
  - For any state s_t, $a_t = \pi(s_t)$ is the action to be taken in that state.
  - Stochastic policy can also be used, which is modeled by a probability.
- Value (or utility) function of state s_t based on policy π:

$$V^{\pi}(s_t)$$

  which is the expected cumulative reward that will be received when the agent follows the policy π, starting from state s_t (a.k.a. V-function).

Finite-Horizon and Infinite-Horizon Models

- The agent tries to maximize the value.
- Finite-horizon model:

$$V^\pi(s_t) = E[r_{t+1} + r_{t+2} + \cdots + r_{t+T} \mid s_t] = E\left[\sum_{i=1}^{T} r_{t+i} \mid s_t\right]$$

- Infinite-horizon model:

$$V^\pi(s_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} \mid s_t\right]$$

where $0 \leq \gamma < 1$ is the discount rate to keep the expected cumulative reward finite.

Value of State vs. Value of State-Action Pair

- Instead of working with the value of state, some applications may prefer working with the value function of state-action pair $Q^{\pi}(s_t, a_t)$ (a.k.a. Q-function).
- For example, for infinite-horizon model:

$$Q^{\pi}(s_t, a_t) = E[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} \mid s_t, a_t]$$

- $V(s_t)$: denotes how good it is for the agent to be in state s_t.
- $Q(s_t, a_t)$: denotes how good it is to perform action a_t in state s_t.
- Optimal policy π^*:

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t), \ \forall s_t$$

- $Q^*(s_t, a_t)$: the value (i.e., expected cumulative reward) of action a_t taken in state s_t and then obeying the optimal policy afterwards, a.k.a. Q-value.

Optimal Policy - I

- For all s_t,

$$\begin{aligned} V^*(s_t) = \max_\pi V^\pi(s_t) &= \max_\pi E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} \mid s_t\right] \\ &= \max_\pi E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1} \mid s_t\right] \\ &= \max_{a_t} E\left[r_{t+1} + \gamma \max_\pi V^\pi(s_{t+1}) \mid s_t, a_t\right] \\ &= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t, a_t\right] \end{aligned}$$

- Bellman equation:

$$V^*(s_t) = \max_{a_t}\left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1})\right)$$

a necessary condition for optimality associated with the mathematical optimization method known as dynamic programming.

Optimal Policy - II

- The value of a state following π^* is equal to the value of the best possible action

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$

Then, we can also write

$$\begin{aligned} Q^*(s_t, a_t) =& E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \\ =& E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

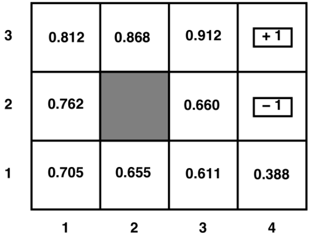
- The optimal policy $\pi^*(s_t)$ can also be defined as:

$$\text{Choose } a_t^* \quad \text{where } Q^*(s_t, a_t^*) = \max_{a_t} Q^*(s_t, a_t)$$

- If we have the $Q^*(s_t, a_t)$ values, then by using a greedy search at each local step we get the optimal sequence of steps that maximizes the cumulative reward.

Utilities of States for Toy Problem

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



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

Outline

Model-Based Learning

- Assume that the environment model parameters $p(r_{t+1} \mid s_t, a_t)$ and $P(s_{t+1} \mid s_t, a_t)$ are known.
- Dynamic programming can be used to find the optimal value function and policy.
- The optimal value function, which is unique, is the solution to the simultaneous equations given by Bellman equation

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t) = \max_{a_t} \Big(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \Big)$$

- Optimal policy:

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1} \in \mathcal{S}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \right)$$

which chooses the action that maximizes the values in the next state.

Value Iteration - I

- If there are n possible states, then there are n Bellman equations, one for each state.
- The n equations contain n unknowns — the utilities of the states.
- So we would like to solve these simultaneous equations to find the utilities.
- There is one problem: the equations are nonlinear, because the “max” operator is not a linear operator.
- Whereas systems of linear equations can be solved quickly using linear algebra techniques, systems of nonlinear equations are more problematic.

Value Iteration - II

- Value iteration is an iterative algorithm for finding the optimal value function and hence the optimal policy.
- It converges to the correct V^* values.
- Convergence criterion:

$$\max_{s \in \mathcal{S}} \left| V^{(\ell+1)}(s) - V^{(\ell)}(s) \right| < \delta$$

  The maximum value difference between two consecutive iterations is below a threshold δ.
- It is possible that the policy converges to the optimal one even before the values converge to their optimal values.

Value Iteration Algorithm

Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s, a) \leftarrow E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V(s')$
        $V(s) \leftarrow \max_a Q(s, a)$
Until $V(s)$ converge

Policy Iteration

- In policy iteration, we store and update the policy rather than doing this indirectly over the values.
- The idea is to start with a policy and improve it repeatedly until there is no change.
- The value function can be calculated by solving the linear equations.
- In each iteration, the policy iteration algorithm has higher complexity than the value iteration algorithm, but policy iteration needs fewer iterations than value iteration.

Policy Iteration Algorithm

Initialize a policy π' arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using π by
        solving the linear equations
            $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V^{\pi}(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s'))$
Until $\pi = \pi'$

Outline

Model-Free Learning

- Sometimes model-based learning may not be the way to go, because:
  - It assumes that the environment model parameters $p(r_{t+1} \mid s_t, a_t)$ and $P(s_{t+1} \mid s_t, a_t)$ are known, but we seldom have such perfect knowledge of the environment in many real applications.
  - Dynamic programming methods are costly.
- Model-free learning:
  - Perfect knowledge of the environment is not available and hence exploration is needed to query the model.
  - The environment is assumed to be stationary.
  - Algorithms can be developed for both deterministic and nondeterministic cases.
- For now, we assume that all $Q(s, a)$ values are stored in a table; we will see later on how we can store this information more succinctly when $|\mathcal{S}|$ and $|\mathcal{A}|$ are large.
- Temporal difference (TD) algorithms:
  - The environment is explored to see the value of the next state and reward.
  - Learning is based on the difference between the current estimate of the value of a state (or a state-action pair) and the discounted value of the next state and the reward received.

Exploration

- ϵ-greedy search:
  - Explore: with probability ϵ, we choose an action uniformly randomly from all possible actions.
  - Exploit: with probability $1 - \epsilon$, we choose the best action so far.
  - We start with a higher ϵ value and then gradually decrease it after we have enough exploration.
- Another exploration strategy is, in state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$ is chosen probabilistically using the softmax function with probability:

$$P(a \mid s) = \frac{\exp[Q(s, a)/T]}{\sum_{b \in \mathcal{A}} \exp[Q(s, b)/T]}$$

  where T is a temperature parameter which is large (to favor exploration) in the beginning and decreases gradually — a process referred to as annealing.

Deterministic Rewards and Actions

- At any state-action pair, there is a single reward and next state possible.
- The equation

$$Q^*(s_t, a_t) = E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

  reduces to

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

- Updating $Q(s_t, a_t)$: in state s_t, a stochastic strategy is used to choose action a_t which returns a reward r_{t+1} and moves to state s_{t+1}

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

- Backup: the later value $\hat{Q}(s_{t+1}, a_{t+1})$, which has a higher chance of being correct, is discounted by γ and added to the immediate reward r_{t+1} (if any) to give the new estimate for $\hat{Q}(s_t, a_t)$.

Nondeterministic Rewards and Actions

- The uncertainty in the system may be due to factors that we cannot control in the environment.
- We have to work with this general form:

$$Q^*(s_t, a_t) = E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

- Q-learning:

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta\Big(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)\Big)$$

- The values $r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$ can be thought of as a sample of instances for each (s_t, a_t) pair. We would like $\hat{Q}(s_t, a_t)$ to converge to the mean of the sample.
- With η gradually decreased in time, the Q-learning algorithm converges to the optimal Q^* values.

Q-Learning Algorithm

Initialize all $Q(s,a)$ arbitrarily
For all episodes
    Initalize s
    Repeat
        Choose a using policy derived from Q, e.g., ϵ-greedy
        Take action a, observe r and s'
        Update $Q(s,a)$:

$$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

        $s \leftarrow s'$
    Until s is terminal state

Temporal Difference Learning

- The Q-learning algorithm is a temporal difference algorithm where we look at the difference between our current estimate of the value of a state-action pair $Q(s, a)$ and the discounted value of the next state-action pair and the reward received.
- The same idea of temporal difference can also be used to learn $V(s)$ values, instead of $Q(s, a)$. Temporal difference (TD) learning uses the following update rule to update a state value:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \eta\Big(r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)\Big)$$

Eligibility Trace

- The Q-learning algorithm is one-step, i.e., the temporal difference is used to update only the previous value of the state or state-action pair.
- An eligibility trace is a record of the occurrence of past visits and enables us to implement temporal credit assignment to update the values of previously occurring visits as well.
- The Q-learning algorithm can be extended to make use of an eligibility trace, giving the $Q(\lambda)$ algorithm.

Outline

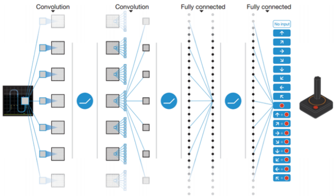
From Q-Learning to DQN - I

- Until now, we assumed that the $Q(s, a)$ values (or $V(s)$, if we are estimating values of states) are stored in a lookup table, and the algorithms we considered earlier are called tabular algorithms.
- Problems with this approach:
  - when the number of states in $\mathcal{S}$ and the number of actions in $\mathcal{A}(s_t)$ is large, the size of the table may become quite large;
  - states and actions may be continuous, for example, turning the steering wheel by a certain angle, and to use a table, they should be discretized which may cause error;
  - when the search space is large, too many episodes may be needed to fill in all the entries of the table with acceptable accuracy.
- Q-learning has no prediction ability, and hence no generalization ability.
- Instead of storing the Q values as they are, we can consider this as a regression model characterized by θ, i.e.,

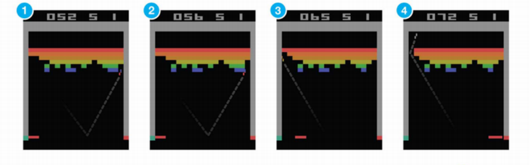
$$Q^*(s_t, a_t \mid \theta) \approx Q^*(s_t, a_t)$$

From Q-Learning to DQN - II

- Regression models can be linear and nonlinear, and we can get rid of the feature engineering process by using the deep neural networks.
- Deep Q-networks (DQN) approximates a state-value function, i.e., Q-values, in the Q-learning framework with a neural network.

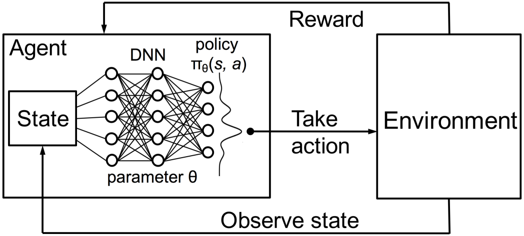


- In the Atari Games case, they take in several frames of the game as an input and output state values for each action as an output.



Deep Reinforcement Learning

- Deep reinforcement learning (DRL): a combination of reinforcement learning (for problem formulation) and deep learning (for $\mathcal{S}$ and $\mathcal{A}(s_t)$ modeling)



- Due to the successfulness in areas of deep supervised learning, more and more models for DRL are coming out...
- DRL could constitute a solution to artificial general intelligence (AGI).