Deep Learning
Reinforcement Learning

dsai.asia

Asian Data Science and Artificial Intelligence Master's Program

Readings

Readings for these lecture notes:

- Ng, A. (2017), *Reinforcement Learning and Control*. Lecture note set 12 for CS229, Stanford University.
- Sutton, R.S. and Barto, A.G. (2018), *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press.

Outline

Introduction

Now we turn to recently developed methods in the area of reinforcement learning.

First, a short review of tabular RL methods.

Then we'll dive into the more recently developed deep learning methods for RL.

In the supervised learning setting, we give the machine examples of what it should do in particular circumstances.

Sometimes it is not easy or even possible to come up with such a training set. Examples include robot motion control, game playing systems, stock trading policies, dynamic allocation of resources to a Web application, ...

In some of these domains, rather than providing a definite answer, we provide the machine with a reward function indicating when it is doing well and when it is doing poorly.

Example reward functions:

- Robot control: give positive rewards for moving forward and negative rewards for falling over.
- Game playing: give positive rewards for winning, negative rewards for losing.
- Robotic exploration: give positive rewards for expanding the "frontier," negative rewards for going over the same old ground.
- Robotic vacuum cleaning: give positive rewards for cleaning dirty areas, negative rewards for spending time in clean areas.
- Web application resource allocation: positive rewards for serving requests quickly, negative rewards for using excessive resources.

Usually, reinforcement learning problems are posed as Markov decision processes.

Outline

A Markov decision process is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$, where

- S is a set of states.
- A is a set of actions.
- P_{sa} are the state transition probabilities. For each state $s \in S$ and action $a \in A$ P_{sa} is a distribution over S. $P_{s_i a_j}(s_k)$ gives the probability that we transition to state s_k when we perform action a_j in state s_i.
- $\gamma \in [0, 1)$ is called the discount factor.
- $R : S \times A \to \mathbb{R}$ is the reward function. Sometimes we use $R : S \to \mathbb{R}$.

Markov decision processes

Payoff

A MDP proceeds as follows:

- Start in some state s_0.
- Choose action $a_0 \in A$.
- Sample $s_1 \sim P_{s_0 a_0}$.
- Choose action a_1.
- Sample $s_2 \sim P_{s_1 a_1}$.
- Repeat...

We can represent the progress of the process as

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots$$

Given such a sequence, we can compute the total payoff as

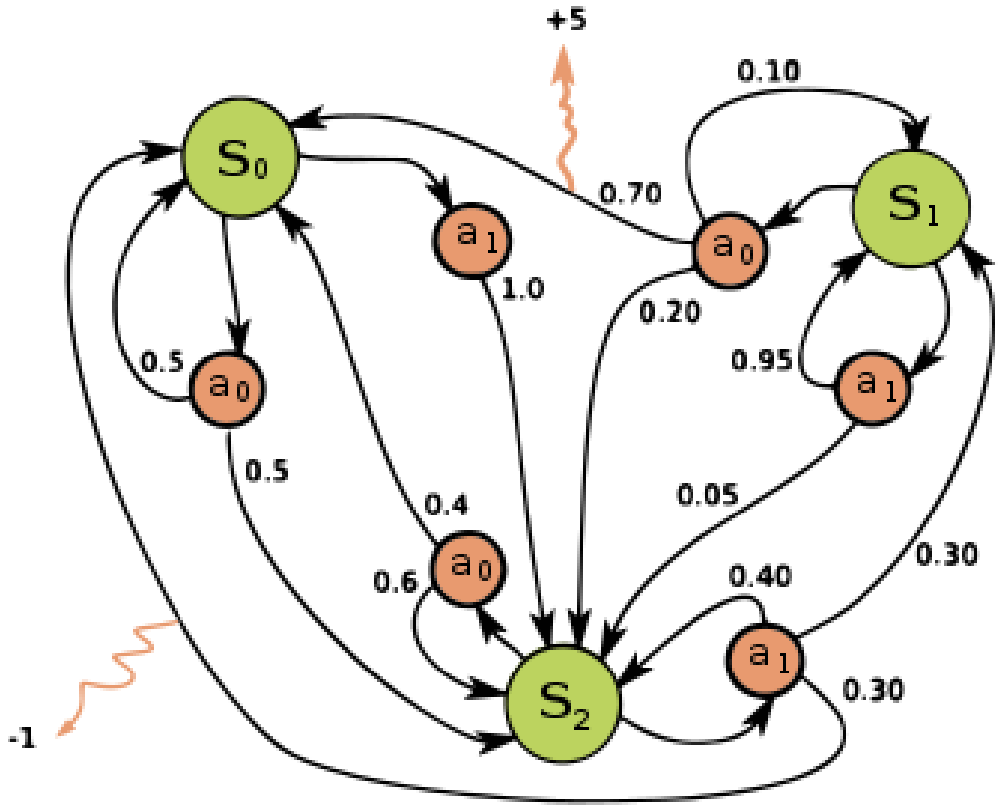
$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

or

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots .$$

Markov decision processes

Simple MDP example



https://en.wikipedia.org/wiki/Markov_decision_process

The reward at timestep t is discounted by a factor γ^t.

In some applications, time might not matter and we would use $\gamma = 1$.

In many other applications, we want to accrue positive rewards as quickly as possible, so $\gamma < 1$.

When $R(\cdot)$ is the amount of money we make, γ would represent the effect of interest (money accrued today is more valuable in the long term than money accrued tomorrow).

How should our learning agent behave?

A policy is a function $\pi : S \to A$ mapping from states to actions.

Executing policy π means when we are in state s, we take action $a = \pi(s)$.

The value function for policy π is

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s; \pi],$$

i.e., the expected sum of discounted rewards upon starting in state s and taking actions according to π.

Given a fixed policy π, the value function V^π satisfies the Bellman equations

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s').$$

$R(s)$ is called the immediate reward we get for starting in state s.

The second term (the summation) can be rewritten $\mathbb{E}_{s' \sim P_{s\pi(s)}}[V^\pi(s')]$, i.e., the expected sum of discounted rewards for starting in state s', where s' is distributed according to $P_{s\pi(s)}$, the distribution over states resulting from action $\pi(s)$ in state s.

The Bellman equations give us a simple way to determine V^π for finite-state MDPs ($|S| < \infty$).

Simply write down the equation $V^\pi(s)$ for each $s \in S$.

When R, P_{sa}, and π are fixed, we have $|S|$ linear equations in $|S|$ unknowns.

[Exercise: find V^π for a small MDP such as a robot maze escape.]

We define the optimal value function as

$$V^*(s) = \max_\pi V^\pi(s).$$

We can write the Bellman equations for the optimal value function as

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s').$$

We have the immediate reward plus the maximum over all a of the expected future sum of discounted rewards we'll get from a.

We define the optimal policy $\pi^* : S \to A$ as

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s') \tag{1}$$

Some facts:

$$\forall s, V^*(s) = V^{\pi^*}(s).$$

$$\forall s, \pi, V^{\pi^*}(s) \geq V^{\pi}(s).$$

i.e., π^* as defined in Equation (1) is the optimal policy.

Note that π^* is optimal for *all* states s. So there is no need to modify π^* according to the start state.

So our question, then, will be how can we find π^* for a given MDP?

So far, we have characterized the value of a state s using the state value function $V^{\pi}(s)$ as the expected sum of discounted rewards we get from executing π from state s.

Many algorithms use, instead of V^{π}, the state-action value function $Q^{\pi}(s, a)$, the expected sum of discounted rewards we get from executing a in s then following π.

$$Q^{\pi}(s,a) = \sum_{s'} P_{sa}(s') \left[R(s,a) + \gamma \sum_{a'} \pi(a' \mid s') Q(s', a') \right]$$

Outline

When S and A are small discrete sets, tabular learning methods are appropriate.

If P_{sa} is also known, we say we have a model of the environment.

We can find the optimal policy for small S and A with known P_{sa} using simple iterative algorithms like value iteration and policy iteration.

Such methods are tabular model based methods, as they use knowledge of P_{sa}.

Usually, P_{sa} is not known.

The simplest method for learning P_{sa} is to add transition learning to the value iteration algorithm.

Among recent deep learning methods for RL, MuZero from DeepMind is an example of a method that learns an explicit representation of P_{sa} during exploration.

Can we learn π^* without knowing or learning P_{sa}?

Methods that do this are called model free.

Monte Carlo methods are methods in which the agent learns about V^* or Q^* by

1. Selecting action a in state s according to π
2. Executing action a leading to s'
3. Repeat 1 and 2 until end of the episode
4. Use the rewards obtained during the episode to update Q^π or V^π and π.

Monte Carlo methods tend to be slow because they don't update π until the episode is finished.

Temporal difference methods update estimates of V^π or Q^π iteratively based on other learned estimates, as the method is executing.

SARSA and Q-learning are popular tabular model free TD methods.

SARSA is a simple method for finding π^*.

It is an on-policy TD policy iteration method.

The SARSA update is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right].$$

The method is called SARSA due to the quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

Algorithm SARSA ON-POLICY TD CONTROL

Input: learning rate α, exploration probability ε
$\forall s, a$, initialize $Q(s, a)$ arbitrarily except $Q(\text{terminal}, \cdot) = 0$
For each episode:
  Initialize s, choose a using policy derived from Q (e.g., ε-greedy)
  For each step in episode:
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$
    $s \leftarrow s'; a \leftarrow a'$

There's a nice example of SARSA for the "Windy Gridworld" example in Sutton and Barto on p. 130.

Exercise: reproduce the SARSA result shown in the graph on p. 130, then do Exercise 6.9 on p. 131.

Q-learning (Watkins, 1989) was one of the early breakthroughs in RL.

It is also fundamental to the Mnih et al. (2013) breakthrough with DQN.

The Q-learning update is

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right].$$

Compare with SARSA.

Actions are taken using ε-greedy, but the updates are based on the action that would be taken with the optimal policy, without ε-greedy.

Because of this difference, Q-learning is an off-policy TD method.

Algorithm Q-LEARNING OFF-POLICY TD CONTROL

Input: learning rate α, exploration probability ε
$\forall s, a$, initialize $Q(s, a)$ arbitrarily except $Q(\text{terminal}, \cdot) = 0$
For each episode:
    Initialize s
    For each step in episode:
        Choose a using policy derived from Q (e.g., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$

Note that Q-learning is a little simpler than SARSA, and note precisely why it is off-policy.

There is a nice demonstration of the differences between SARSA and Q-learning, the "Cliff Walking" example in Sutton and Barto on p. 132.

Exercise: implement SARSA and Q-learning for the cliff walking example to demonstrate that Q-learning learns an optimal policy that is different from (and better than) the ε-greedy policy learned by SARSA.

The TD control algorithms involve maximization during policy construction.

Q-learning's target policy is the greedy policy, which involves taking action a maximizing $Q(s, a)$.

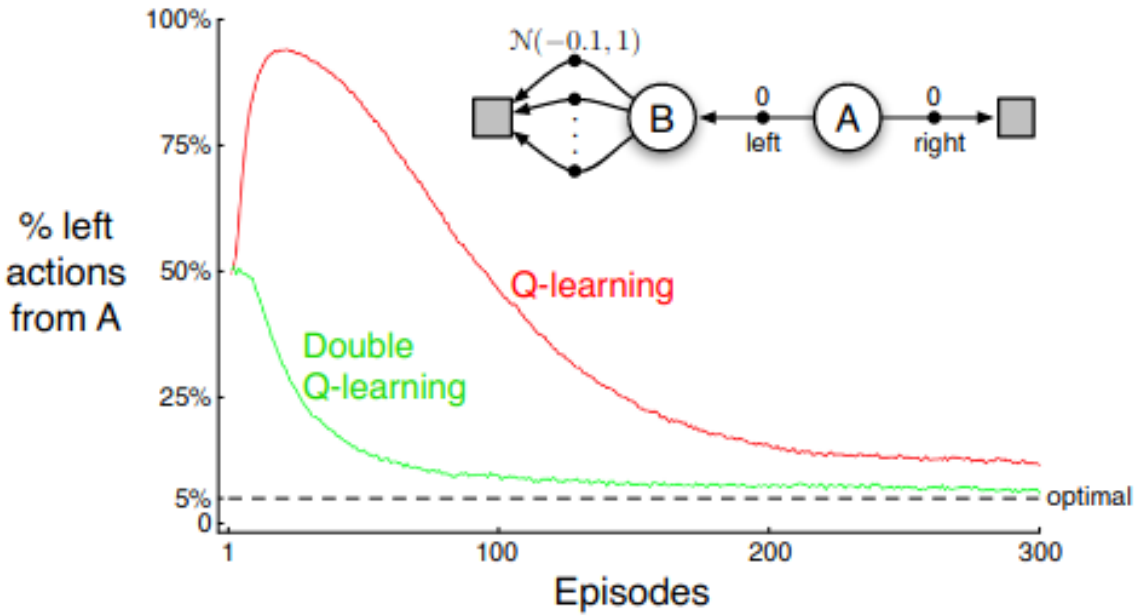
SARSA's policy is, for example, ε-greedy, also involving maximization.

The max operator tends to introduce bias into the value estimates.

Consider a state s for which all actions a have 0 value. With uncertain but unbiased estimates of $Q(s, a)$, some will be positive and some will be negative.

$\max_a Q(s, a)$, on the other hand, will almost always be positive, a positive bias called maximization bias.

Sutton and Barto give an example MDP in which the optimal action is always to move "right" from state A, but due to maximization bias and the high variance of the distribution over rewards, Q-learning will overestimate the value of a "left" move.



Sutton and Barto (2018), Fig. 6.5

Idea to learn without maximization bias: divide the episodes into two sets and use them to learn two separate estimates Q_1 and Q_2 of $Q(a)$.

We use Q_1 for maximization:

$$a^* = \underset{a}{\text{argmax}}\, Q_1(a),$$

and we use Q_2 as the estimate of the value:

$$Q_2(a^*) = Q_2(\underset{a}{\text{argmax}}\, Q_1(a)).$$

The expectation

$$\mathbb{E}\left[Q_2(a^*)\right] = q(a^*)$$

is unbiased.

We also use Q_1 as an unbiased estimate of the value of actions selected using Q_2:

$$Q_1(a^*) = Q_1(\operatorname{argmax}_a Q_2(a)).$$

This is called double learning.

We learn two estimates, but only one is updated on each episode.

Double Q-learning uses this idea to get more accurate, less biased estimates of values and can learn an optimal policy for the left/right environment above.

Algorithm DOUBLE Q-LEARNING for $Q_1 \approx Q_2 \approx q$

Input: learning rate α, exploration probability ε

$\forall s, a$, initialize $Q_1(s, a)$ and $Q_2(s, a)$ arbitrarily except $Q(\text{terminal}, \cdot) = 0$

For each episode:

Initialize s

For each step in episode:

Choose a using ε-greedy with $Q_1 + Q_2$

Take action a, observe r, s'

With probability $\frac{1}{2}$:

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha \left[r + \gamma Q_2(s', \text{argmax}_{a'} Q_1(s', a')) - Q_1(s, a) \right]$$

else:

$$Q_2(s, a) \leftarrow Q_2(s, a) + \alpha \left[r + \gamma Q_1(s', \text{argmax}_{a'} Q_2(s', a')) - Q_2(s, a) \right]$$

$s \leftarrow s'$

We've covered MDPs, the idea of Monte Carlo control, and two of the most popular model-free tabular methods for RL, SARSA and Q-learning.

These are the foundations of Deep Reinforcement Learning.

Outline

Continuous state MDPs

Examples

Now that we can estimate models for P_{sa} and R, we have the following remaining assumptions:

- $|S| < \infty$
- $|A| < \infty$
- The outcome s' obtained from executing action a in state s is observable.
- The reward $R(s)$ obtained from state s is immediately observable.

What about relaxing the finite state assumption?

In a car or ground robot, we might have a state $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta})$. $S = \mathbb{R}^6$.

In the inverted pendulum problem, we have $(x, \theta, \dot{x}, \dot{\theta})$. $S = \mathbb{R}^4$.

For an aircraft, we have $(x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi})$. $S = \mathbb{R}^{12}$.

Now tabular learning is impossible, and we need deep RL methods!

Outline

Partially observable Markov decision processes (POMDPs)

Sometimes we are not able to fully observe s_t.

Instead, we may only be able to take an observation related to s_t, such taking a picture with a camera or a distance scan with a laser or a GPS reading from a GPS device.

This leads to the idea of the partially observable Markov decision process (POMDP).

We assume that at each timestep t we obtain an observation $z_t \in \mathcal{O}$ related to the state s_t by a distribution $P_{zs}(z \mid s)$.

A POMDP with finite horizon is thus given by a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, P_{sa}, P_{zs}, T, R)$.

There are many methods for dealing with POMDPs. The simplest approach, used in most deep RL methods, is to simply treat z_t as the state s_t!

Outline

Conclusion

Now we understand the foundations of reinforcement learning and are ready to tackle modern deep RL algorithms.