S&DS 265 / 565
Introductory Machine Learning

Reinforcement Learning

Tuesday, November 30

Yale

The home stretch...

13	Nov 19-28	No class, Thanksgiving break			
14	Nov 30, Dec 2	Reinforcement learning	🔗 Reinforcement learning	Tue: Assn 6 in; Assn 7 out	Nov 30: Reinforcement learning
15	Dec 7, 9	Societal issues for machine learning		Tue: Quiz 4 Thu: Assn 7 in	

The home stretch...

- Assignment 6 due today
- Assignment 7 (neural nets) posted today
- Quiz 4: Week from today (neural nets)
- Two more topics: RL and societal issues
- Panel discussion next week!
- Final exam, Dec 21 at 7pm (cumulative, cheat sheet, practice exam)

For today and Thursday

- Overview of RL

- Q-learning

- Illustration on taxi problem

- Deep reinforcement learning

- Neuroscience connection

Reinforcement learning

- An agent interacts with an environment

- The actions the agent takes change the state of the environment

- The agent receives rewards for each action, and seeks to maximize the total cumulative reward

Reinforcement learning is a framework for sequential decision making to achieve a long-term goal.

Reinforcement learning: Motivating examples

- Playing chess, backgammon, Atari games, etc.
- Learning to walk or ride a bike
- A robot vacuum cleaning up the house
- Preparing a meal

Reinforcement learning: Formalization

- The environment is in state s at a given time
- The agent takes action a
- The environment transitions to state $s' = \text{next}(s, a)$
- The agent receives reward $r = \text{reward}(s, a)$
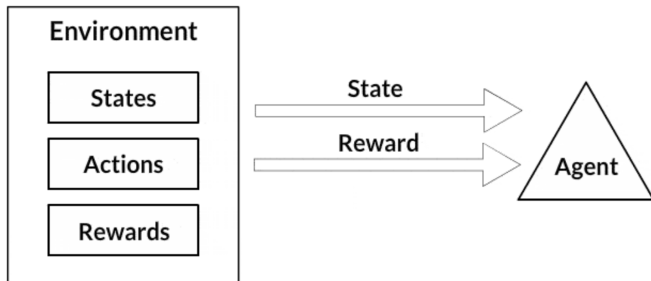
Reinforcement learning: Formalization

- The environment is in state s at a given time

- The agent takes action a

- The environment transitions to state $s' = \text{next}(s, a)$

- The agent receives reward $r = \text{reward}(s, a)$

This is said to be a *Markov decision process*. It's "Markov" because the next state only depends on the current state and the action selected. It's a "decision process" because the agent is making choices of actions in a sequential manner.

Reinforcement learning

Environment
- States
- Actions
- Rewards

Action ← Agent
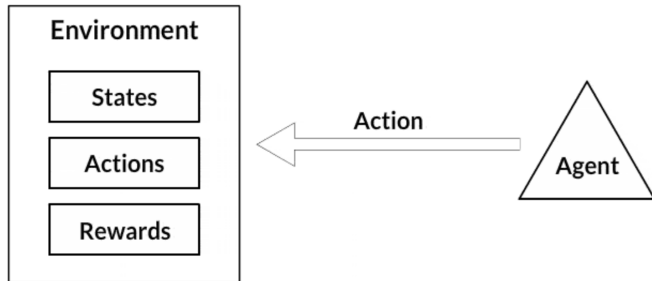
Reinforcement learning

Characteristics

- RL is inherently sequential

- In between supervised and unsupervised learning

- Agent can't act too greedily; needs to be strategic

The aim of RL is to learn to make optimal decisions from experience

Taxi problem

We illustrate this with a toy problem: The Taxi problem from "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition", by Tom Dietterich.

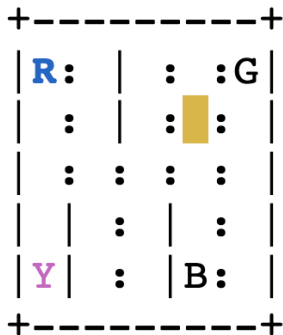The code uses OpenAI gym.

Taxi problem

A taxicab drives around the environment, picking up and delivering a passenger at four locations
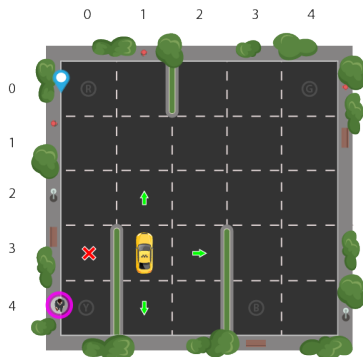
Taxi problem

A taxicab drives around the environment, picking up and delivering a passenger at four locations

"Ascii art" rendition:

Taxi problem: Description

- Four designated locations: R(ed), G(reen), Y(ellow), and B(lue)
- Taxi starts off at random square and passenger is at random location
- Taxi drives to passenger's location, picks up the passenger, drives to passenger's destination, drops off passenger
- Once the passenger is dropped off, the episode ends.

Taxi problem: State space

- 25 taxi positions
- 5 possible locations of passenger: At waiting location or in taxi
- 4 possible destination locations
- Total number of states: $25 \times 5 \times 4 = 500$.

Taxi problem: State space

Passenger location coded as integers:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)
- 4: in taxi

Taxi problem: State space

Destinations coded as:

- 0: R(ed)
- 1: G(reen)
- 2: Y(ellow)
- 3: B(lue)

Taxi problem: State space

Agent actions coded as:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Taxi problem: State space

Rewards:

- Default reward per step: -1
- Reward for delivering passenger: +20
- Illegal "pickup" or "drop-off": -10

Taxi problem: State space

State space represented as a tuple:
   state = (taxi row, taxi column, passenger location, destination)

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s
- Quality should not be assessed purely based on the reward the action has in the current time step

Q-learning

- Maintains a "quality" variable $Q(s, a)$ for taking action a in state s
- A measure of the cumulative rewards obtained by the algorithm when it takes action a in state s
- Quality should not be assessed purely based on the reward the action has in the current time step
- Need to take into account the future rewards

Q-learning update

Update:

$$Q(s, a) \longleftarrow (1 - \alpha)\, Q(s, a) + \alpha\Big(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') \Big)$$

Q-learning update

Update:

$$Q(s, a) \longleftarrow (1 - \alpha)\, Q(s, a) + \alpha\Big(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a')\Big)$$

- When action a is taken in state s, reward reward(s, a) is given
- Then, the algorithm moves to a new state next(s, a)

Q-learning update

Update:

$$Q(s, a) \longleftarrow (1 - \alpha)\, Q(s, a) + \alpha \Big(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') \Big)$$

- For example, if the taxi is location $(2, 2)$ and takes the "West" action ($a = 3$), then there is a reward of -1, and the taxi moves to the new location $(2, 3)$

- If cab is empty, it remains empty, and if it contains the passenger, the passenger remains.

Q-learning update

Update:

$$Q(s, a) \longleftarrow (1 - \alpha)\, Q(s, a) + \alpha \Big(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') \Big)$$

- Cumulative future reward of this action is $\max_{a'} Q(\text{next}(s, a), a')$
- Future rewards discounted by factor $\gamma < 1$
- Trades off short-term against long-term rewards

Q-learning update

Better written as

Update:

$$Q(s, a) \longleftarrow$$
$$Q(s, a) + \alpha \Big(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \Big)$$

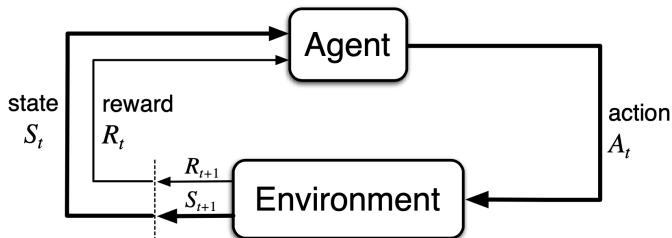Viewed as gradient *ascent* algorithm, with step size α

Let's go to the notebook!

Principles of RL

Now that we've seen it in action, let's introduce some key RL concepts and principles:

Policy, reward signal, value function, model, Bellman equation

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutten and Andrew G. Barto

Principles of RL

Principles of RL

Policy: A mapping from states to actions. An algorithm/rule to make decisions at each time step, designed to maximize the long term reward.

Principles of RL

Reward signal: The sequence of rewards received at each time step. An abstraction of "pleasure" (positive reward) and "pain" (negative reward) in animal behavior.

Principles of RL

Value function: A mapping from states to total reward. The total reward the agent can expect to accumulate in the future, starting from that state.

Rewards are short term. Values are predictions of future rewards.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutten and Andrew G. Barto

Principles of RL

Model: Used for planning to mimic the behavior of the environment, to predict rewards and next states.

A *model-free* approach directly estimates a value function, without modeling the environment.

"Reinforcement Learning: An Introduction" (Second Edition), Richard S. Sutten and Andrew G. Barto

Example: Tic-Tac-Toe

```
 O |   |
---+---+---
 X | X | O
---+---+---
 O |   | X
```

- Can be approached using *Q*-learning or "value iteration"

- Number of states?

- Unlike classical game (minimax) search, does not assume opponent is playing optimally. Also does not require specification of the opponent.

- Can be learned using "self-play"

Learning strategy: Tic-Tac-Toe

- Set up a table of numbers, one for each possible state
- Each number is latest estimate of the probability of winning from that state
- The table is the value function
- Playing as X, for all states with three Xs in a row, the value is one (similarly 0 for three Os in a row)
- Initialize all other values to $\frac{1}{2}$

Learning strategy: Tic-Tac-Toe

- Set up a table of numbers, one for each possible state

- Each number is latest estimate of the probability of winning from that state

- The table is the value function

- Playing as X, for all states with three Xs in a row, the value is one (similarly 0 for three Os in a row)
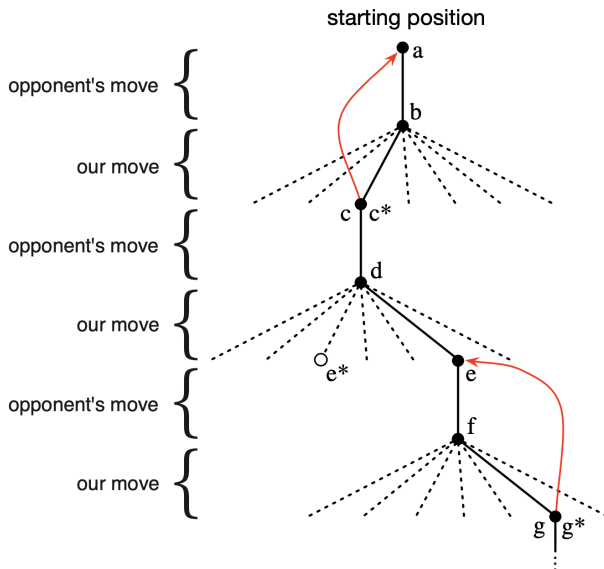
- Initialize all other values to $\frac{1}{2}$

Learning strategy: Tic-Tac-Toe

- Play many games against the opponent

- Most of the time we move greedily, selecting the move that leads to the state with greatest value

- Occasionally, select randomly from the other moves "Exploratory moves"

- For greedy moves, adjust state's value a little bit toward the value of the next state:

$$V(S_t) \longleftarrow V(S_t) + \alpha \left(V(S_{t+1}) - V(S_t) \right)$$

Learning strategy: Tic-Tac-Toe

starting position

opponent's move

our move

opponent's move

our move

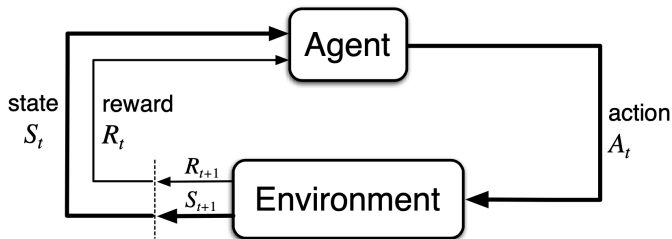opponent's move

our move

a
b
c c*
d
e*
e
f
g g*

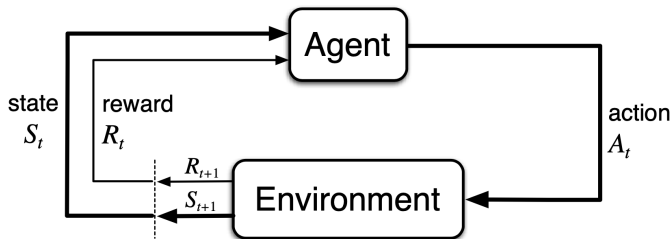# Learning (and unlearning) to ride a bike

---

"Learning to Drive a Bicycle Using Reinforcement Learning and Shaping," Jette Randløv and Preben Alstrøm, ICML 1998
Learning to ride a backwards bike: `https://www.youtube.com/watch?v=MFzDaBzBlL0`

Recall the setup

state S_t | reward R_t | action A_t

R_{t+1}
S_{t+1}

Agent

Environment

Recall the setup

Rewards and state transitions are probabilistic, in general

Bellman equation

The optimal value function is the largest expected discounted long term reward starting from that state.

Bellman equation: Deterministic case

The optimality condition for the value function v_* is

$$v_*(s) = \max_a \left\{ \text{reward}(s, a) + \gamma v_*(\text{next}(s, a)) \right\}$$

Bellman equation: Deterministic case

The optimality condition for the Q-function q_* is

$$q_*(s, a) = \text{reward}(s, a) + \gamma \max_{a'} q_*(\text{next}(s, a), a')$$

and then $v_*(s) = \max_{a'} q_*(s, a')$

Q-learning update

Note how this makes sense in terms of the update rule:

$$Q(s, a) \longleftarrow$$
$$Q(s, a) + \alpha \left(\text{reward}(s, a) + \gamma \max_{a'} Q(\text{next}(s, a), a') - Q(s, a) \right)$$

Bellman equation: Random environments

The optimality condition for the value function v_* is

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\Big\{r + \gamma v_*(s')\Big\}$$

$$= \max_a \mathbb{E}\Big[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a\Big]$$

Bellman equation: Random environments

The optimality condition for the Q-function q_* is

$$q_*(s, a) = \sum_{s', r} p(s', r \mid s, a) \left\{ r + \gamma \max_{a'} q_*(s', a') \right\}$$

$$= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a \right]$$

Summary

- Reinforcement learning is a framework for sequential decision making to achieve a long-term goal

- The agent receives rewards for each action, and seeks to maximize the total cumulative reward

- The value of a state is the total reward the agent can expect to accumulate in the future, starting from that state

- Q-learning is an iterative algorithm that maximizes the "quality" of each state-action pair

- The Bellman equations are optimality conditions that characterize the value function