Reinforcement Learning

Optimization Methods in Management Science
Master in Management
HEC Lausanne

Dr. Rodrigue Oeuvray

Fall 2019 Semester

Introduction

- Many of us have probably heard of AI learning to play computer games on their own, a very popular example being **Deepmind**
- Deepmind hit the news when their **AlphaGo** program defeated the South Korean **Go** world champion in 2016
- AlphaGo follows a paradigm of Machine Learning known as **Reinforcement Learning**

Reinforcement Learning

- **Reinforcement learning** involves an **agent**, a set of **states**, and a set of **actions** per state
- By performing an action, the agent transitions from state to state
- Executing an action in a specific state provides the agent with a **reward** (a numerical score)
- The goal of the agent is to **maximize its total (future) reward**
- It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward
- This potential reward is a weighted sum of the **expected values** of the rewards of all future steps starting from the current state

Reinforcement Learning Analogy

Consider the scenario of training an horse. It doesn't understand our language, so we can't tell him what to do. Instead, we follow a different strategy. We emulate a situation, and the horse tries to respond in many different ways. If the horse's response is the desired one, we reward it with snacks. The next time the horse is exposed to the same situation, it is likely that it will execute a similar action with even more enthusiasm in expectation of more food. That's like learning "what to do" from positive experiences. Similarly, horses will tend to learn what not to do when face with negative experiences

Reinforcement Learning (Cont'd)

Reinforcement Learning lies between the spectrum of **Supervised Learning** and **Unsupervised Learning**. A few things to note :

- **Quick win strategy does not always work**
  - There are things that are easy to do for instant gratification, and there's things that provide long term rewards
  - The goal is to not be greedy by looking for the quick immediate rewards, but instead to optimize for maximum rewards over the whole training
- **Sequence matters in Reinforcement Learning**
  - The reward agent does not just depend on the current state, but the entire history of states
  - Unlike supervised and unsupervised learning, time is important here

The Reinforcement Learning Process

Simply stated, Reinforcement Learning is the science of making **optimal decisions using experiences**. Breaking it down, the process of reinforcement learning involves these simple steps :

1. Observation of the environment
2. Deciding how to act using some strategy
3. Acting accordingly
4. Receiving a reward or penalty
5. Learning from the experiences and refining our strategy
6. Iterate until an optimal strategy is found

Q-Learning

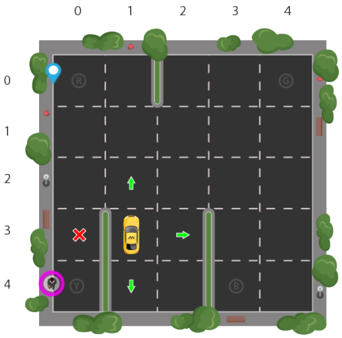
- Q-Learning is a reinforcement learning technique used in machine learning
- The goal of Q-Learning is to learn a **policy**, which tells an agent what action to take under what circumstances
- For any **finite Markov decision process** (FMDP), Q-Learning finds a policy that is **optimal** in the sense that it **maximizes the expected value of the total reward** over any and all successive steps, starting from the current state

Q-Learning Example : The Taxi Problem

- One of the problems from **Gym**
- Gym is a toolkit for developing and comparing reinforcement learning algorithms
- It supports teaching agents everything from walking to playing games like Pong or Pinball
- This problem will be used in the practice session

Problem Description

- There are four designated locations in the grid world indicated by R, B, G, and Y
- When the episode starts, the taxi starts off at a random square and the passenger is at a random location (R, B, G, or Y)
- The taxi drive to the passenger's location, pick up the passenger, drive to the passenger's destination (another one of the four specified locations), and then drop off the passenger
- Once the passenger is dropped off, the episode ends



Observations

- There are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is the taxi), and 4 destination locations
- This makes a total of 500 discrete states

Actions

There are 6 discrete deterministic actions :

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

Rewards

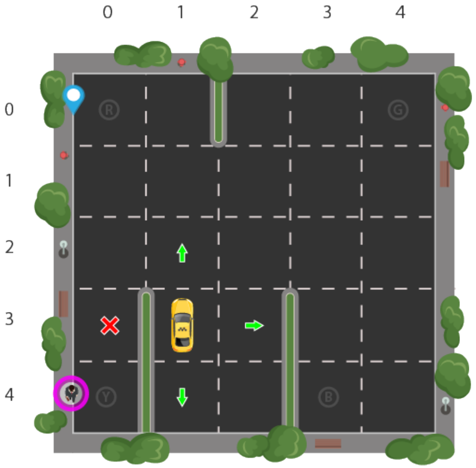
- There is a reward of -1 for each action and an additional reward of +20 for delievering the passenger
- There is a reward of -10 for executing actions "pickup" and "dropoff" illegally

Rendering

- blue : passenger
- magenta : destination
- yellow : empty taxi
- green : full taxi
- other letters : location

Obstacles

They are also obstacles preventing the taxi from doing some moves. If that taxi wants to performs a “forbidden” moves, then it stays at its current location and receive a penality of -1



Introducing the Q-Table

- **Q-table** is the name for a simple lookup table where we calculate the **maximum expected future rewards** for action at each state
- In the Q-table, the columns are the **actions** and the rows are the **states** the cab will get if it takes that action at that state
- This is an **iterative** process, as we need to improve the Q-table at each iteration

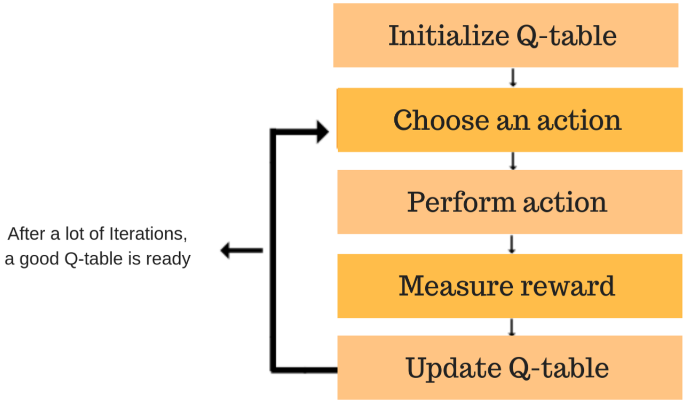
The Q-Learning Algorithm

- The **Q-function** takes two inputs : state (s) and action (a)

$$Q(s_t, a_t) = E[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots | s_t, a_t]$$

- γ is a discount factor ($\leq$1)
- Using the above function, we get the values of Q for the cells in the table
- This is an iterative process of updating the values
- As we start to explore the environment, the Q-function gives us better and better approximations by continuously updating the Q-values in the table

The Q-Learning Algorithm Process



The Q-Learning Episode

- The agent will learn through experience, without a teacher (unsupervised learning)
- It will explore from state to state until it reaches the goal
- Each exploration is called an **episode**
- Each episode consists of the agent moving from the initial state to the goal state
- Each time the agent arrives at the goal state, the program goes to the next episode

The Q-Learning Algorithm

Step 0. Initialize the Q-table

Step 1. For each episode :

1.1 Select a random initial state

1.2 Do while the goal state hasn't been reached

- Select one among all possible actions for the current state
- Using this possible action, consider going to the next state
- Get maximum Q-value for this next state based on all possible actions
- Update the Q-table
- Set the next state as the current state

Bellman Equation

- The main concept behind Q-learning is the Bellman equation :

$$Q(s,a) = r + \gamma max_{a'} Q(s', a')$$

- **This equation states that the Q-value for a certain state-action (s, a) pair should be the reward r gained by the moving to the new state s' added to the value of the best action in the next state**

- A reward right now is more valuable than receiving a reward on the future, and that's why we have a discount factor γ

Update Q-table

The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information :

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

where r_t is the reward received when moving from the state s_t to the state s_{t+1}, α is the learning rate $0 < \alpha \leq 1$, and γ is the discount factor

The Learning Rate

- The **learning rate** or step size determines to what extent newly acquired information overrides old information
- A factor of 0 makes the agent learn nothing (exclusively exploiting prior knowledge), while a factor of 1 makes the agent consider only the most recent information (ignoring prior knowledge to explore possibilities)
- In fully deterministic environments, a learning rate of $\alpha_t = 1$ is optimal
- When the problem is stochastic, the algorithm **converges** under some technical conditions on the learning rate that require it to decrease to **zero**
- In practice, often a constant learning rate is used, such as $\alpha_t = 0.1$ for all t

The Discount Factor

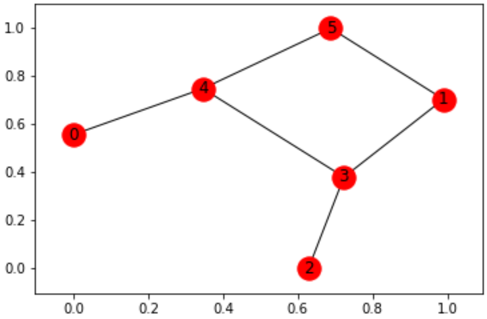
- The **discount factor** $0 \leq \gamma \leq 1$ determines the importance of future rewards
- A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, i.e. r_t, while a factor approaching 1 will make it strive for a long-term high reward
- For $\gamma = 1$, without a terminal state, or if the agent never reaches one, all environment histories become infinitely long, and utilities with additive, undiscounted rewards generally become **infinite**

Updating the Q-table : Example

- Suppose we have 5 rooms in a building with some of them connected
- We'll number each room from 0 to 4
- The outside of the building can be thought of as one big room numbered 5
- Notice that doors 1 and 4 lead into the building from room 5 (outside)
- The plan of the room can be represented by a graph

Graph of the Rooms

Representation of the rooms as a graph. Node 5 is the **goal** state



Goal of the Experiment

- Imagine our agent as a dumb virtual robot that can learn through experience
- The agent can pass from one room to another but has no knowledge of the environment, and doesn't know which sequence of doors lead to the outside
- **We would like that the robot learn how to reach outside the building**

Example (Cont'd)

To understand how the Q-learning algorithm works, we'll go through a few episodes step by step

- Initialize Q-table as a zero 6×6 matrix
- Define the reward matrix R defined by $R(i,j) =$ reward if we apply action j to state i. Action j results in reaching state j

$$R = \begin{pmatrix} \color{red}{0} & \color{red}{0} & \color{red}{0} & \color{red}{0} & 0 & \color{red}{0} \\ \color{red}{0} & \color{red}{0} & \color{red}{0} & 0 & \color{red}{0} & 10 \\ \color{red}{0} & \color{red}{0} & \color{red}{0} & 0 & \color{red}{0} & \color{red}{0} \\ \color{red}{0} & 0 & 0 & \color{red}{0} & 0 & \color{red}{0} \\ 0 & \color{red}{0} & \color{red}{0} & 0 & \color{red}{0} & 10 \\ \color{red}{0} & 0 & \color{red}{0} & \color{red}{0} & 0 & 10 \end{pmatrix}$$

- When $R(i,j) = \color{red}{0}$ (in red), it means that it is not possible to apply action j to state i. In that case, the agent stays at the same state

Example (Cont'd)

- In this example, we will use the following (simplified) recursive formula to update the Q-table :

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \times max_a Q(s_{t+1}, a)$$

- Let's assume that the initial state is 1 with $\gamma = 0.8$
- Then, several moves are possible. We select action 5 to go to state 5
- Update of the Q-table :

$$Q(1,5) = R(1,5) + 0.8 \times max_a Q(5,a) = 10 + 0.8 \times 0 = 10$$

Example (Cont'd)

- The next state, 5, now becomes the current state
- Because 5 is the goal state, we've finished one episode
- The updated Q-table is given by :

$$Q\text{-table} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example (Cont'd)

- For the next episode, we choose state 3 as our initial state
- Let's assume that we go to state 1 from state 3
- Then :

$$Q(3,1) = R(3,1) + 0.8 \times max_a Q(1,a) = 0 + 0.8 \times 10 = 8$$

Example (Cont'd)

- The next state, 1, now becomes the current state. We go to state 5
- Then :

$$Q(1,5) = R(1,5) + 0.8 \times max_a Q(5,a) = 10 + 0.8 \times 0 = 10$$

Example (Cont'd)

- The episode is over
- The updated Q-table is given by :

$$Q\text{-table} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Example (Cont'd)

After performing a few episodes, we may obtain the following Q-table :

$$Q\text{-table} = \begin{pmatrix} 0 & 0 & 0 & 0 & 40 & 0 \\ 0 & 0 & 0 & 32 & 0 & 50 \\ 0 & 0 & 0 & 32 & 0 & 0 \\ 0 & 40 & 25.6 & 0 & 40 & 0 \\ 32 & 0 & 0 & 32 & 0 & 50 \\ 0 & 40 & 0 & 0 & 40 & 50 \end{pmatrix}$$

Example (Cont'd)

- Once the Q-table gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state
- Tracing the best sequences of states is as simple as following the links with the highest values at each state

Example (Cont'd)

- Example : from the initial state 2, the Q matrix tells us to go state 3 for a reward of 32, then to go to state 1 or 4 for a reward of 40 and finally to reach state 5 for a reward of 50

$$Q\text{-table} = \begin{pmatrix} 0 & 0 & 0 & 0 & 40 & 0 \\ 0 & 0 & 0 & 32 & 0 & 50 \\ 0 & 0 & 0 & 32 & 0 & 0 \\ 0 & 40 & 25.6 & 0 & 40 & 0 \\ 32 & 0 & 0 & 32 & 0 & 50 \\ 0 & 40 & 0 & 0 & 40 & 50 \end{pmatrix}$$

- **We conclude that the robot has learned from experience how to reach the outside of the building !**

Exploration vs Exploitation

- Online decision making involves a fundamental choice :
  - Exploitation : make the best decision given current information
  - Exploration : gather more information
- The best long-term strategy may involve short-term sacrifices
- Gather enough information to make the best overall decisions

Examples

- Restaurant Selection
  - Exploitation : Go to favorite 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
- Clinical Trial
  - Exploitation : Choose the best treatment so far
  - Exploration : Try a new treatment

ϵ-Greedy Algorithm

- This is one of the simplest possible algorithms for trading off exploration and exploitation, where you make the decision at each step of the way
- You can either choose to take a random action or choose the agent's trusted action
- This algorithm more or less mimics the greedy algorithm as it generally exploits the best available option, but sometimes explores the other available options

ϵ-Greedy Algorithm (Cont'd)

The next action a_{t+1} is simply chosen as :

$$a_{t+1} = \begin{cases} a^*_{t+1} & \text{with prob. } 1-\epsilon, \\ \text{random action} & \text{with prob. } \epsilon, \end{cases}$$

where a^*_{t+1} is the optimal action and $0 \leq \epsilon \leq 1$

The Initial Conditions Q_0

- Since Q-learning is an iterative algorithm, it implicitly assumes an initial condition before the first update occurs
- The most basic approach consists in initializing this matrix with zeros
- High initial values, also known as *optimistic initial conditions*, can encourage exploration : no matter what action is selected, the update rule will cause it to have lower values than the other alternative, thus increasing their choice probability
- Concretely, with high initial values, the reward is less important that these values which will drive the selection of the next action

The Initial Conditions Q_0 (Cont'd)

- The first reward r can be used to reset the initial conditions. According to this idea, the first time an action is taken the reward is used to set the value of Q
- This allows immediate learning in case of fixed deterministic rewards. A model that incorporates **reset of initial conditions** (RIC) is expected to predict participants' behavior better than a model that assumes any **arbitrary initial condition** (AIC)