University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 24: November 15

Lecturer: Alexander Schwing Scribe: Haojia Yu

1 Overview

The first part of the lecture reviewed one reinforcement learning model which is covered in last lecture, Markov Decision Process. How to define a Markov decision process? The four ingredients of Markov decision process. The three ways to define the best policy $\pi*$: Exhausted search, Policy iteration and Value iteration. How to compute V$*$, Q$*$, $\pi*$ and evaluate fixed policy $\pi*$ using policy/value iteration or exhaustive.

The second part of the lecture introduced the topic on Q-learning. Q-learning is a reinforcement learning technique in machine learning, which does not require the transition probabilities. This part of lecture introduced the questions followed. Why do we need to use Q-learning and what problem does it solve? How does the Q-learning work and what is the algorithm of the Q-learning. How to evaluate the fixed policy π? The lecture also gave an example of Atari games using q-learning. Deep Q-learning algorithm (Deep Q-networks (DQN)) is also introduced and explained in this part. [4]

2 Review

What is Markov decision process? The three ways to define the best policy $\pi*$: Exhausted search, Policy iteration and Value iteration. How to compute V$*$, Q$*$, $\pi*$ and evaluate fixed policy $\pi*$ using policy/value iteration or exhaustive.

2.1 Markov decision process

What is Markov decision process and how to define it?
It has four ingredients:
States s $\in$ S. Actions a $\in$ A. Transition probability P($s' \mid s, a$). Reward function R(s, a, s').
Given the present state, the future and the past are independent. So the probability that state S is the state after t steps of actions is

$$P(S_t + 1 = s' \mid S_t = s_t, A_t = a_t, S_t - 1 = s_t - 1, ..., S0 = s0) = P(S_t + 1 = s' \mid S_t = s_t, A_t = a_t) \tag{1}$$

Where the it already take t steps by taking action a_i at state s_i and resulting at the state $s_i + 1$ for every steps.

The thing which is wanted to be find using Markov decision process is the policy according to which the actions taken by machine can get the highest future reward. (Policy: Given the state, what action do we need to take) [1].

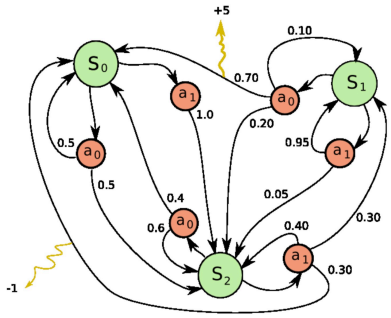

Figure 1: Pictorial representation of MDP:

2.2 The three ways to define the best policy

Exhausted search
All the possibilities of policies need to be know. Define the policy at first.
Care about expected future reward $V^{\pi*}(s_0)$ to evaluate quality of π.
Need to find the policy π^* with the highest expected future reward.

The expected reward obtained equals to the reward when start from a state s and take action a according to the policy π plus the value of state s' obtained by acting according to the policy start from state s'. Since there are more than one possible state it will go to, the expected reward of all possibilities will be multiplied by the expectation (the transition probabilities) and added up to get the future expected reward. This method allowed to get the expected reward according to the policy and find the optimal one.

The disadvantage is that it's very expensive since using the linear equations. Iterative refinement can be used but it's still expensive since it still need to search all the policies.
In Iterative refinement, it initialize the values and update for several iterations. Start from randomly chosen values and update according to the rule to get the V^{π}.

Policy iteration
Policy iteration will not search all the policies but start from some policies and get the optimal one with greedy method. It Choose one policy randomly, repeat the following steps until policy π does not change(You have two successful iterations with the same policy (machine take the same action)):
Get the value of a state according to the policy π

$$V^{\pi}(s) = \sum_{s' \in S} P(s' \mid s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{2}$$

Then change the policy by looking all the actions that we can execute when being in a particular state. We take the action which can give us the best future reward.

$$\pi(s) = \underset{a \in A_s}{\operatorname{argmax}} \sum_{s' \in S} P(s' \mid s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{3}$$

We find the best future reward by getting the reward when start from a state and act according to the policy π plus the value of state s' obtained by acting according to the policy starting from s'.

Start from one policy, get a better policy by getting the future reward of the policy. It's guaranteed

to find the best policy since the second part in the iteration is a greedy method using the previous values of state s'.

Value iteration
Directly find the optimal value V$^{/}ast$ of the value function, then find the policy π which get the best value.

Instead of thinking about the policy space, the searching space need to be considered. Start with the Bellman optimality principle which tell you the optimal value according to the condition of states, actions and transition probabilities. The expected reward will be get in state s when executing the optimal policy. Then we can rewrite the optimality value principle instead of defining it.

$$V^*(s) = \max_{a \in A_s} \sum_{s' \in S} P(s' \mid s, a)[R(s, a, s') + V^*(s')] \tag{4}$$

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a)[R(s, a, s') + \max_{a' \in A_{s'}} Q^*(s')] \tag{5}$$

Different from the V*, the Q* is another function which defined by the state s and the action a. V function tell us the expected value executing action a according to the policy π at state s. Q function tell us the expected value executing action a at state s. It doesn't matter whether you use V or Q. But people usually prefer the Q function since it's easier to decode the policy.

Here are the method to decode the policy according to the two functions S and Q.
V function: Look at all possible actions can be executed, you need to compute the best future reward when performing the action a while being in state s. Take the action which give the largest future reward.

$$\pi(s) = \operatorname*{argmax}_{a \in A_s} \sum_{s' \in S} P(s' \mid s, a)[R(s, a, s') + V^*(s')] \tag{6}$$

Q function: Maximize the Q^* with the respect of a. Choose the best action which give us the largest expected future reward in state s.

$$\pi(s) = \operatorname*{argmax}_{a \in A_s} Q^*(s, a) \tag{7}$$

3 The problem of MDP

In some condition, transition probabilities are not known. The delivered state is not even know. The designer of the algorithm don't know how many states are there and what are they. The state to pass the game is not even known. How can we find the best policy of the agent?

The following is the Bellman optimality principle:

$$Q^*(s, a) = \sum_{s' \in S} P(s' \mid s, a)[R(s, a, s') + \max_{a' \in A_{s'}} Q^*(s')] \tag{8}$$

What can we do to get the optimal policy if we don't know the transition probabilities $P(s' \mid s, a)$?

We need to estimated the transition probabilities by running a simulator which collect the experience tuples/samples (s,a,r,s′). By keeping playing the game, a large dataset of experience tuples/samples can be get. By using the samples, we can approximate the transition probabilities and evaluate the policy. We will have enough guess to the transition probabilities if we have a very large experience tuples/samples. It might not be 100% accurate, but after very long-term training, the accuracy will be close enough to the idea policy which is good enough for usage.
But even we have the transition probabilities and rewards, if we don't know the states or cannot determine what is the state, the task will never be finished. Even it have states, if there are too many of them, it will take forever to finish the task.

To solve the reinforcement learning problems without transition probabilities and rewards ot without model, the Q-learnign is introduced.

4 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. It does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations. [4]

Instead of collecting lots of transition probabilities, Q-learning just collect few of samples of the MDP (In some states, act some actions, get some reward and go to some other states and get some rewards) even it doesn't know the model and transition probabilities, but it's very sure about the (s,a,r,s′). Then it can obtain the sample suggests by using Q function which will be introduced later.

4.1 Algorithm sketch

We assume that the function $Q(s, a)$ should be approximately equal to the function $y_{(s,a,r,s')}$ which is determined by the samples. The y equals to the reward r plus maximizing all actions which can be executed in the next state s'.

$$Q(s, a) \approx y_{(s,a,r,s')} = r + \max_{a' \in A_{s'}} Q(s', a')] \tag{9}$$

So it won't need to take a long time to get the transition probabilities and use the approximation calculate the optimal Q function. But it will get the samples and get calculate the optimal Q function at the same time. To get the accurate expectation of reward plus maximum Q for state s', the agent will keep trying the same action from state s to get the reward and the state s' for so many times. Then, by simulating these times, the agent can get the approximate probability of the state s and action a. The main idea is doing thing jointly instead of sequentially.
To get the accurate missing transition probability, the agent keep updating the $Q(s, a)$ according to the the following formula:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha y_{(s,a,r,s')} \tag{10}$$

What it does is using the current estimated Q function and update the Q value of the state s, action a to when trying to move it into the value which y function suggested. With large amount of samples. the Q function will compromise with all the y function taken from lots of (s, a, r, s'). It will be closer to the optimal Q function.
The function (10) can be written in this way:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a' \in A_{s'}} Q(s', a') - Q(s, a)] \tag{11}$$

We need to notice the learning rate α which means how important is the new sample in training. If the value is larger, the new Q function will be more close to the new sample and less close to the old Q function. Tuning to get a good α value is a very important thing when building a model using Q-learning.

4.2 The steps of Q-learning

Step 1: Initialize Q-values We build a Q-table, with m cols (m= number of actions), and n rows (n = number of states). Initialize the values at 0.

Repeat followed steps until the training reached a maximum number of episodes or the training is stopped manually.

Step 2: Choose the action Choose an action a in the current state s based on the current Q-value estimates.

There is also a method to optimize the action selection, the exploration rate "epsilon" which is used to change strategies of choosing the actions. Every iteration, we need to generate a random number. If this number is larger than epsilon, then we will do just use what we already know to select the best action at each step. Else, explore more possibilities of actions.

At the beginning, we will set "epsilon" value to its highest value 1 because we don't know anything about the true values of Q-table. We need to do a lot of exploration and change the Q function with more respect to the samples gotten by randomly choosing actions. We need to make sure the agent explore more possibilities of actions at the start.

Then the agent need to reduce the value of"epsilon" and becomes more confident about the Q-table it already got for estimating Q-values.

Step 3: Evaluate Take the action a and observe the state "s'" it goes and reward r. Then update the function Q(s,a). Then, to update $Q(s, a)$ we use the Bellman equation which is the formula (11).

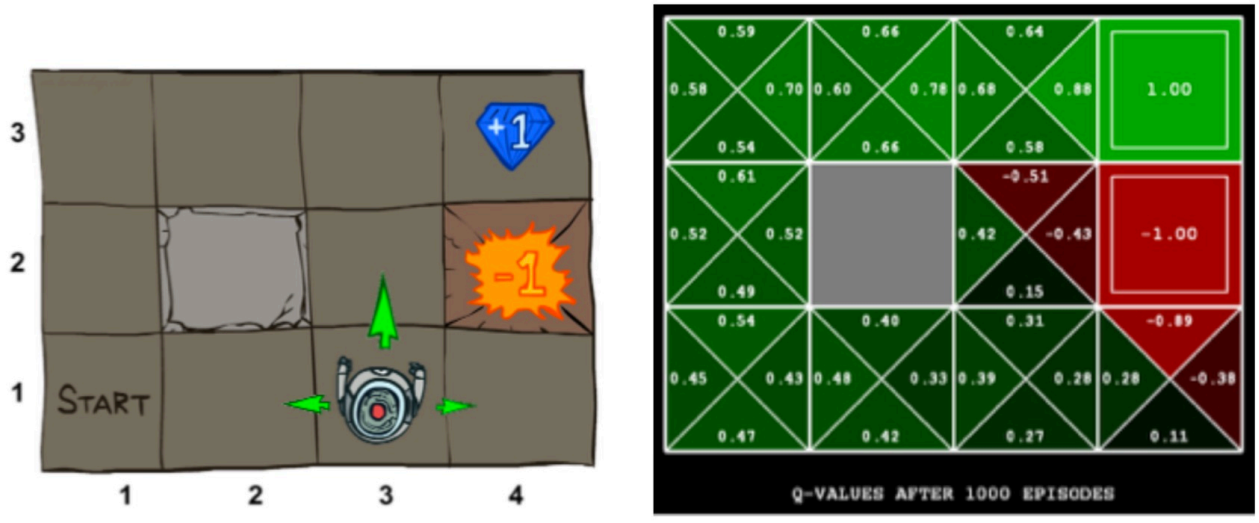

Figure 2: Tabular game with Q-learning

In the Figure 2, we have an agent which can move on the board in four directions. it board have

one point with +1 reward and another point with -1 reward. We can run this agent on the board over and over again and update the Q values. We can see from the figure that in every state, it have four Q value for four different actions (Move up, move down, move right and move left).

The policy gotten will be based on the maximum value of the Q function in every state. Choose the action with the maximum Q value. The following figure will show how the agent will go according to the current policy.

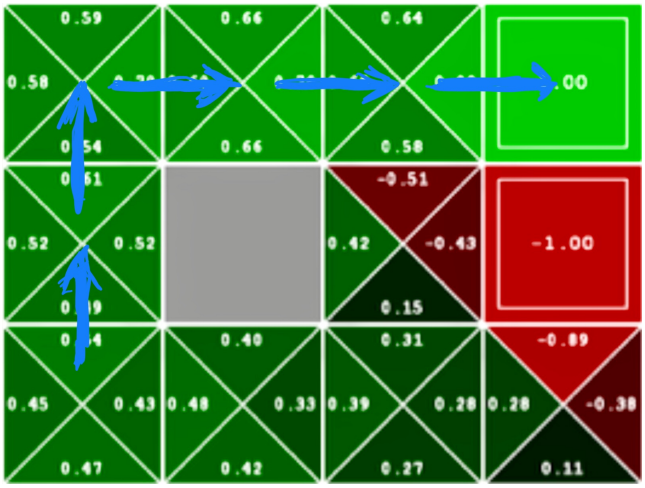

Figure 3: How agent will move given Q values

4.3 MDP and Q-learning

From the information of last and this lecture:
If we know the MDP, we use policy iteration to calculate the $V*$, $Q*$, $\pi*$ and using policy/value iteration or exhaustive search to evaluate fixed policy $\pi*$. If we don't know the MDP, we use Q-learning to calculate the $V*$, $Q*$, $\pi*$ and using value learning to evaluate fixed policy $\pi*$.

4.4 Questions which is more complex

What if we don't know what actions, rewards or even the states of a MDP?
For example, the Atari games. We know the actions (Move up, move down, move right and move left) and the reward (Reward is positive when reaching the destination, negative when meet with some moving object. And something else). But we don't know the state of game since the map of the game will change in different round. There is no model!

We need a function $Q_\theta(s, a)$ as a neural net which takes images as input.

5 Deep Q-networks (DQN)

Deep Q-Networks (DQN) is a kind of network using q-learning algorithm referring to convolutional networks. When training the agent, the input will go into the network and the weight will be changed. So the Q value of all possible actions will be gotten according to the the input.

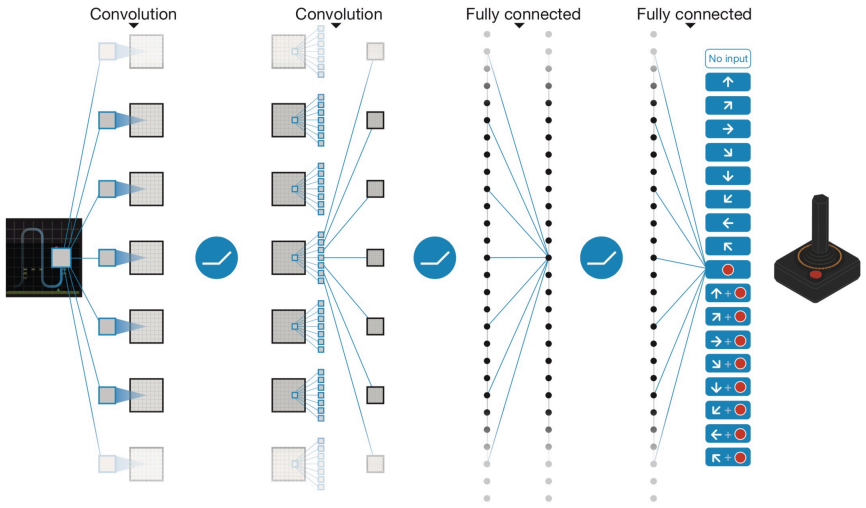

Figure 4: Deep Q-Networks (DQN)

5.1 The algorithm of DQN

The Q value of all possible actions according to the input will be stored as real number. As the figure shown above, the network will be trained with the optimal weight and Q value of all possible actions according to the input if the training data set is big enough.

Given dataset $D = \{(s_j, a_j, r_j, s_{j+1})\}$.
randomly grab a minibatch $B \subseteq D$.
Compute the y function which is mentioned on Q-learning.

$$y_j = r_j + \gamma \max_a Q_\theta{}^-(s_{j+!}, a) \tag{12}$$

Use stochastic semi-gradient descent to optimize with respect to the parameters θ.

$$\min_\theta \sum_{s_j, a_j, r_j, s_{j+1} \in B} (Q_\theta(s_j, a_j) - y_j)^2 \tag{13}$$

So the $Q_\theta(s, a)$ should be close enough to the y_j. Use the *min* to make the y value not depended on θ. When computing the gradient, it will depend on the dataset D. It is very like the deep learning network which get the imput of image, forward path and get the action. Compare the Q with y_j and square as loss function. Then the network can back update the weights to get the network better and better. After performing the update, use the new Q function to execute another step, the environment will give out a data point. Add the data point to dataset and keep running in this way. [2]

Perform $\epsilon-$greedy action and augment D. The $\epsilon-$greedy action is mentions in the previous section. In every iteration, we need to generate a random number. If this number is larger than epsilon, then we will do just use what we already know to select the best action at each step. Else, explore more possibilities of actions. The $\epsilon-$greedy action algorithm will set the ϵ to its highest value 1 at the start to avoid the training from being dominated by the Q function. Then it will decrease the value since it's confident enough about the model has been trained and trust more on the Q function and rely on the training less.

There are many important things to make sure the training getting the optimal policy. We need to make sure going through all over the space of possibilities of actions, states. Another thing which is important to Q-learning is making the reward is big enough to make sure the Q-learning or DQN go to the right direction during training.

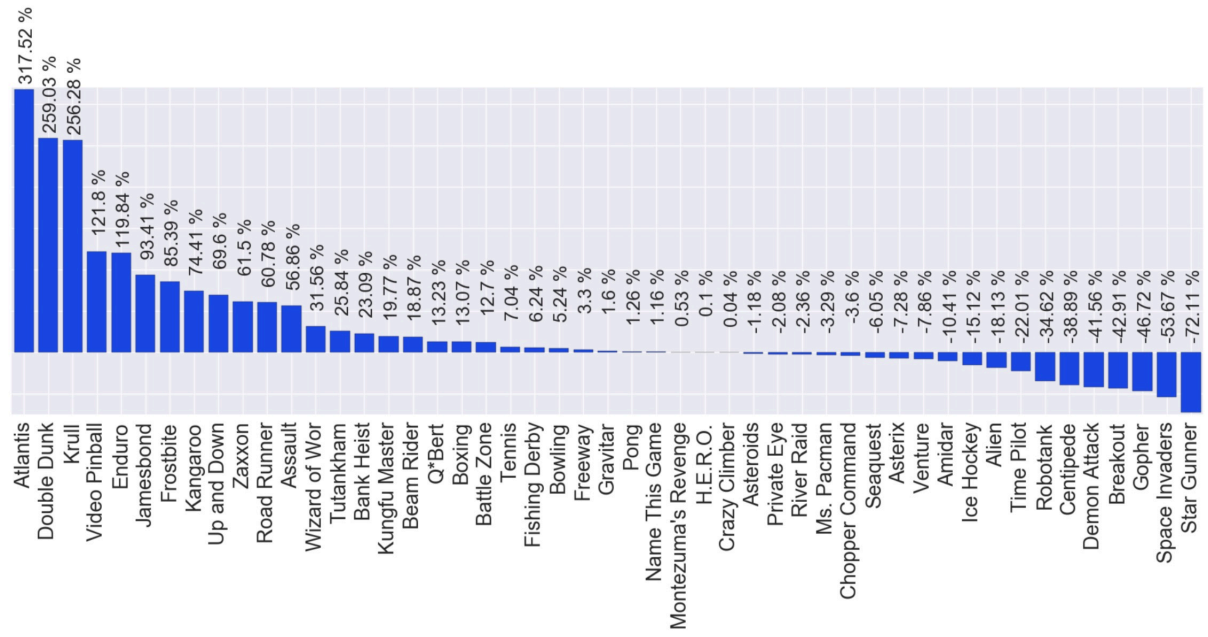

Figure 5: The result of playing different Atari games with DQN

6 Quiz

6.1 What differentiates RL from supervised learning?

Supervised learning:

Given a bunch of input data X and labels Y. During the training, it learned the function f: X $\rightarrow$ Y which maps X to Y. After the training process, the function will be able to predict Y from input data of X format with a certain accuracy.
Test process is about a chunk of data the model has never seen.
And it's used to classify labels or to produce real numbers.

Reinforcement Learning:

Input 5-tuple: states, actions, reward, transaction probability, start and terminal states. Output: Many possible solutions.
We are given a set of states S, a set of actions A and P which is the state transition probability. The reward tells us how good we did in terms of the goal we want to optimize towards. According to this value during the training, we n get the optimal policy. The process continues and the model keeps on learning.
The task is to learn the policy function π which maps from states to actions. The process need to find an optimal policy that maximizes rewards.
And it's used to find an optimal policy which maximizes the reward for the model/agent.

6.2 What is a MDP?

A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning. [3]

It has four ingredients:
States s ∈ S. Actions a ∈ A. Transition probability P($s' \mid s, a$). Reward function R(s, a, s').
Given the present state, the future and the past are independent. So the probability that state S is the state after t steps of actions is

$$P(S_t + 1 = s' \mid S_t = s_t, A_t = a_t, S_t - 1 = s_t - 1, ..., S0 = s0) = P(S_t + 1 = s' \mid S_t = s_t, A_t = a_t) \tag{14}$$

6.3 What to do if no transition probabilities are available?

Approximating the policy by samples. Use the Q-learning algorithm to get if we don;t know the states. Using the current estimated Q function and update the Q value of the state s, action a to when trying to move it into the value which y function suggested.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[3] Wikipedia. *Markov decision process.* `https://en.wikipedia.org/wiki/Markov_decision_process`.

[4] Wikipedia. *Q-learning.* `https://en.wikipedia.org/wiki/Q-learning`.