Abstract—Q-learning is a machine learning algorithm that aims to solve a general Markov Decision Problem without knowing its model beforehand. This algorithm uses dynamic programming algorithm as its core element and can be extended with an artificial neural network to solve an infinite and indefinite Markov Decision Problem

Keywords—Markov, dynamic programming, machine learning, artificial intelligence, reinforcement learning, Q-learning

I. INTRODUCTION TO MARKOV DECISION PROCESS

Many problems that humanity faces in this world often ultimately breaks down to a decision making process. A chess game for example, can be abstracted into a making decision of which piece to move to which square according to a certain rule.

Markov Decision Process is a logical framework to encapsulate and explain a general decision making process. Formally, it is known as a 5-tuple:

\[(S, A, P, R, \gamma)\]

- \(S\) is a finite state of states
- \(A\) is a finite state of actions
- \(P(s, s')\) is the probability that action \(a\) in state \(s\) will lead to state \(s'\) in the next timeslice.
- \(R(s, s')\) is the immediate reward that is achieved after \(s\) transitioned to \(s'\) due to action \(a\)
- \(\gamma\) is the discount factor, which states the importance of future rewards. The smaller the number, the more greedy is the decision process; the more it values immediate rewards.

There are 2 kinds of Markov Decision Process, which is deterministic and stochastic

- Deterministic Markov Decision Process
  Deterministic Markov Decision Process is a Markov Decision Process in which the next state of the decision process is completely determined only by the current state and the action taken at that state. There is no random element that determines the next state; there is no probability function associated with this kind of Markov Decision Process.

- Stochastic Markov Decision Process
  Stochastic Markov Decision Process is a Markov Decision Process in which the next state of the decision process has some element of uncertainty within it. There exist some probability value of achieving a certain state given the current state and what action is being taken.

To solve a Markov Decision Problem is to find a policy function \(\pi(s)\) that will produce the optimal action to choose when in state \(s\); an action that will produce maximum cumulative reward.

Solving a Markov Decision Problem can be divided into 2 categories based on the agent’s (an entity who walks through the decision process) knowledge about the Markov model: model-based approach and model-free approach.

- Model-based approach
  With the model-based approach, the agent already knows the Markov model of the world and simply searching for the optimal policy function.

- Model-free approach
  In the model-free approach, the agent doesn’t have the idea of the world. It doesn’t know the reward function and the probability function. It tries to derive the policy function via trial-and-error while exploring the world.

The model-free approach in Markov Decision Process will be explained further later in the paper. The model-based approach can be divided into 2 class of algorithms: policy iteration algorithms and value iteration algorithms. These 2 algorithm is based on the same equation

\[V^\pi(s) = \sum P_{\pi(s)}(s, s') \cdot (R_{\pi(s)}(s, s') \cdot \gamma V^\pi(s'))\]

- Value iteration algorithm
  In value iteration algorithm, there is a value function that calculates the total reward that an agent will get if
it starts at state $s$ given a policy function. This algorithm will start with a random value function, propagates through the Markov Model until the goal, and then updates its value function. This process will be repeated until it reaches the optimal value function. After an optimal value function is found, the optimal policy function is then extracted from the optimal value function.

- **Policy iteration algorithm**

  Policy iteration algorithm differs from value iteration algorithm in that it will update the policy function after each step of the algorithm, rather than after each cycle, so it can find the optimal policy faster.

Once the policy function has been fixed, the decision making system will behave like a Markov Chain; a sequence of events.

A Markov Decision Process can be represented in many ways. Commonly it is represented by a finite state machine, although this representation only applies to a finite Markov Decision Process.

```
II. INTRODUCTION TO DYNAMIC PROGRAMMING

Dynamic programming is a class of algorithm of solving an optimality problem by dividing it into individual sub-problems and storing the solution of each sub-problems so that it only solved once. This definition of dynamic programming is stated by Richard Ernest Bellman, the creator of dynamic programming.

In practice, this is done by building a solution upon the result of the previous solution; concatenating the individual sub-problems. The problem is divided into many stages, and the solution of a given stage is built upon the solution of the previous stage, recursively.

This is gives birth to the Richard Bellman’s Principle of Optimality:

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision".

Stated another way, if a policy in a given state is optimal, then the solution on the previous stage is also optimal.

This class of algorithm shares a similarity of idea from divide-and-conquer algorithms in that they try to divide the problem into a smaller sub-problems and solve it individually. What differentiate dynamic programming algorithms with divide-and-conquer algorithms is that in dynamic programming, the smaller sub-problems can overlap. Because the individual sub-problems is concatenated and combined into a bigger sub-problems, the bigger sub-problems can share individual sub-problems with one another.

This class of algorithm also shares a similarity of idea with greedy algorithms in that they take only the optimal solution of the next immediate state. What differentiate dynamic programming with greedy algorithm is that dynamic programming takes into consideration all the optimal solution in a given stage. You can think of dynamic programming as a greedy algorithm that runs in parallel.

Dynamic programming can be divided into 2 categories. Those are forward and backward.

- **Forward**

  Finding the optimal solution by setting the starting state as the base of recursion, so that the algorithm propagates forward at searching the next optimal solution.

- **Backward**

  Finding the optimal solution by setting the goal state as the base of recursion, so that the algorithm propagates backward at searching the next optimal solution.

III. INTRODUCTION TO Q-LEARNING

Thinking machine is a thing that has been occupying the mind of many engineers and philosophers even before the dawn of computer science. The idea of a mechanical mind and an artificial intelligence that could rival its biological counterparts is simply intriguing and challenging to explore. One of the artificial intelligence fields that has seen many breakthrough and advancements lately is the field of machine learning.

Machine learning is a field of computer science that is concerned with giving computers the ability to learn to do something, without being explicitly programmed. This is done with the means of inferential statistics; to deduce the underlying property of the computation data. This insight of the underlying property is then used to do something that the user wants the computer to accomplish.

Commonly, machine learning algorithm is divided into 3 subsection, based on the learning process and how the outcome
is used: supervised learning, unsupervised learning, and reinforcement learning.

1. Supervised Learning

Supervised learning is a type of machine learning algorithms that learns from a cluster of correctly-done data. The inferred function is then used to produce an output from new input data.

2. Unsupervised Learning

Unsupervised learning is a type of machine learning algorithms that tries to draw inferences between data without any correctly-done example.

3. Reinforcement Learning

Reinforcement learning is a type of machine learning algorithms that gives an agent reward or punishment based on what action it takes while doing something. This kind of algorithm is inspired by the study of behavioral science of animals.

Reinforcement learning is another word for a model-free approach in Markov Decision Process and is further divided into 2 categories. Those are Monte Carlo methods and temporal difference methods.

1. Monte Carlo methods

Monte Carlo methods is a reinforcement learning method that will learn, evaluate, and adjust its decision process with each full path finished i.e. when it arrives at the goal node. This method would not introduce a bias upon the initial condition of learning parameter, because it learns what the true goal should be. Although unbiased, it only works on episodic and small and finite decision process.

2. Temporal difference methods

Temporal difference methods is a reinforcement learning method that will learn, evaluate, and adjust its decision process with each step in the process. Unlike Monte Carlo methods, temporal difference method will work on an infinite decision process, because it only take concern of the partial decision process. The downside is that temporal difference method will have a biased step updates on the initial condition of the learning parameter.

This paper will take concern on a special kind of temporal difference method of reinforcement learning that’s called Q-learning.

IV. Knowing an Unknown Markov Decision Process through Q-learning

Q-learning is a method to learn the optimal policy function \( \pi(s) \) in a Markov Decision Process (a policy function that will give a maximum cumulative reward in the goal state) given an unknown probability \( P_a(s, s') \) function or unknown reward \( R_a(s, s') \) function. The name comes from the Q function used in this method of learning. Q stands for “quality” and it is a function that measures the maximum possible reward that an agent will get for doing action \( a \) in a state \( s \) that gets transitioned to state \( s' \). For a finite Markov Decision Process, the Q function is basically a table where the column represents the states and the row represents the action.

The Q function is calculated with the Bellman Equation:

\[
Q(s, a) = \sum_{s'} P_a(s, s') \cdot (R_a(s, s') + \gamma \max_{a'} (Q(s', a')))
\]

- \( Q(s, a) \) is the Q function
- \( P_a(s, s') \) is the probability that action \( a \) in state \( s \) will lead to state \( s' \) in the next timeslice.
- \( R_a(s, s') \) is the immediate reward that is achieved after \( s \) transitioned to \( s' \) due to action \( a \)
- \( \gamma \) is the discount factor, which states the importance of future rewards. The smaller the number, the more greedy is the decision process; the more it values immediate rewards.

Because the agent doesn’t know about the probability or the reward function, the Q table is initialized with any random value.

If the probability function is also unknown, it can be approximated separately while the agent is learning.

The agent then simply takes the action which produces the biggest Q value. Stated in the context of Markov Decision Process:

\[
\pi(s) = \text{argmax} (Q(s, a))
\]

The agent learn through a state-action pair and updates its Q table after each action is executed. This is done using the following formula:

\[
Q_{\text{new}}(s, a) = (1 - \alpha) \cdot Q_{\text{old}}(s, a) + \alpha \cdot Q_{\text{observed}}(s, a)
\]
α is the learning rate and it goes from 0 to 1. The bigger the value, the more aggressive it will learn and the lower the value, the more conservative the agent is.

However, in the beginning the agent doesn’t know anything about the decision process, so the initial value that is produced by the Q function is not a valid basis for determining an action. We can remove this tendency of adhering to the initial Q function by introducing a new probability constant ε, the exploration constant.

With probability of ε, π(s) will return a random action a and with probability of 1−ε, π(s) will take the learned solution via the solution written above. Overtime, this constant is slowly reduced until it reaches zero, signaling that the agent have explored many possible actions and has learned the Markov Decision Process.

V. DYNAMIC PROGRAMMING IN Q-LEARNING

The Q function described by the Bellman Equation implemented the principles of backward dynamic programming.

The function produces a value that is the sum of the immediate reward and the optimal value of the Q function in the next stage (the max a).

By the principle of optimality, it will also give an optimal value of the Q function if we take the maximum value over the stage.

VI. EXAMPLE OF Q-LEARNING IN PATH FINDING AI

Suppose that there is a path finding game with a map as given:

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- State 1 is the start state
- State 11 is the goal state
- R(1, up) = -0.1
- R(1, right) = -0.1
- R(2, left) = -0.1
- R(2, right) = -0.1
- R(3, left) = -0.1
- R(3, up) = -0.1
- R(3, right) = -0.1
- R(4, left) = -0.1
- R(4, up) = -5
- R(5, up) = -0.1
- R(5, down) = -0.1
- R(6, up) = -0.1
- R(6, down) = -0.1
- R(6, right) = -5
- R(7, up) = 5
- R(7, down) = -0.1
- R(7, left) = -0.1
- R(8, right) = -0.1
- R(8, down) = -0.1
- R(9, left) = -0.1
- R(9, right) = -0.1
- R(10, left) = -0.1
- R(10, down) = -0.1
- R(10, right) = 5
- R(11, left) = -0.1, R(11, down) = -5
- A deterministic model, if the action is up, it will go up 100% of the time

The Q-learning algorithm will go as follows:

1. Initialize Q function, α, and γ.

The Q function will be represented as a table, because the problem is a finite Markov Decision Problem. We will initialize the Q function randomly.

<table>
<thead>
<tr>
<th></th>
<th>left</th>
<th>down</th>
<th>up</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-999</td>
<td>-999</td>
<td>0.15</td>
<td>0.30</td>
</tr>
<tr>
<td>2</td>
<td>-0.83</td>
<td>-999</td>
<td>-999</td>
<td>0.08</td>
</tr>
<tr>
<td>3</td>
<td>1.42</td>
<td>-999</td>
<td>0.94</td>
<td>-3.24</td>
</tr>
<tr>
<td>4</td>
<td>2.22</td>
<td>-999</td>
<td>1.09</td>
<td>-999</td>
</tr>
<tr>
<td>5</td>
<td>-999</td>
<td>1.98</td>
<td>-1.89</td>
<td>-999</td>
</tr>
<tr>
<td>6</td>
<td>-999</td>
<td>0.88</td>
<td>-0.48</td>
<td>-0.11</td>
</tr>
<tr>
<td>7</td>
<td>1.01</td>
<td>0.02</td>
<td>-0.78</td>
<td>-999</td>
</tr>
<tr>
<td>8</td>
<td>-999</td>
<td>-1.88</td>
<td>-999</td>
<td>1.23</td>
</tr>
<tr>
<td>9</td>
<td>0.94</td>
<td>-999</td>
<td>-999</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>0.56</td>
<td>1.01</td>
<td>-999</td>
<td>-0.72</td>
</tr>
<tr>
<td>11</td>
<td>-1.99</td>
<td>2.10</td>
<td>-999</td>
<td>-999</td>
</tr>
</tbody>
</table>

α and γ is arbitrarily chosen to be 0.5
2. Initialize $\varepsilon$ and its decay factor

We will initialize the $\varepsilon$ with 0.7 and the decay factor of 0.001 that is, the $\varepsilon$ will decrease 0.001 each turn. Thus the learning process will take 700 epoch.

3. Pick an action and update the current state

Based on the $\varepsilon$ we take, there are 0.7 chance that we will take a random action in this state and 0.3 chance that we will take a calculated action in this state.

Suppose that we take a calculated action. We will take the ‘right’ action, since it produces the greatest (argmax) Q value (0.30)

If the updated current state is the goal state, move the agent to the start state.

4. Observe the reward and updates Q function

We will then update the Q function with the observed reward (-0.1) and the maximum possible Q value on that state (0.08). With the Bellman Equation:

$$Q(1, \text{right}) = 0.5 \cdot 0.3 + 0.5 \cdot (-0.1 + 0.5 \cdot 0.08)$$

$Q(1, \text{right})$ is now 0.12

5. Decrease $\varepsilon$

$\varepsilon$ is now 0.699

6. Repeat step 3, 4, 5 until $\varepsilon$ reaches 0

After 700 cycles of learning, the optimal path from the learning will either be these two:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>^</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Or:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>^</td>
</tr>
</tbody>
</table>

As these two path will result in the same cumulative reward, which is 4.6

VII. THE FUTURE: Q-LEARNING ON AN INFINITE AND INDEFINITE MARKOV DECISION PROCESS

Many interesting problems in the real world is not as simple as the example problem shown above. Problems like chess, buying and selling stocks, playing video games, and building self-driving cars doesn’t have a finite and definite decision process.

An infinite and indefinite Markov Decision Process can’t be modeled with a Q table. It has too many possibilities of state-action pairs to enumerate and to take actions with.

In 2013, a group of researchers at DeepMind, a British AI startup company, has successfully built a machine learning model that harvests input straight from sensory data. They built an AI that can play general Atari 2600 games and beats human expert. The input is not the internal game state but instead, raw pixels data outputed from the screen.

They accomplish this through combining Q-learning with deep convolutional neural network. Essentially, they replace the Q table with a neural network whose input is the raw pixels of the game and the action and whose output is the estimated future rewards.

This research has took humanity a step towards the holy grail of artificial intelligence, that is a general artificial intelligence. One that could solve any complex problem that is given to it.
VIII. CONCLUSION

Q-learning is a useful and efficient method to be used on a Markov Decision Process without knowing the decision process model beforehand, like many problems that is present on computer science and engineering in general.

THANK YOU NOTE

This paper would not be possible without the help of many contributing agents. Thank you to Mr. Rinaldi Munir, Mrs. Masayu Leylia Khodra, and Mrs. Nur Ulfa Mauliadevi for sharing their knowledge in the Algorithm Strategy course.

REFERENCES


PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Mei 2018

S.S. Diwangkara 13516107