An Improved Q-Learning Algorithm and its Application to a Real Time Resource Management Game

Seungwhan Moon¹, Paul Booth¹, Mina Choe²
¹ Franklin W. Olin College of Engineering; Needham, MA; U.S.A.
² Korea Advanced Institute of Science and Technology; Daejeon; Republic of Korea

Abstract

This article addresses an improved Q-learning algorithm that learns the desired behavior more quickly and efficiently than a classic Q-learning algorithm. When a reward is given to a state, the new Q-learning algorithm updates the Q-value of not only the current state but also the Q-values of its adjacent states. This algorithm is suitable for a certain game that has a large number of states of the same kind (i.e., amount of a certain resource, a health level, etc.). As such, a new resource management game was designed as an example of its implementation based on the new Q-learning algorithm. In addition, a test bed was designed in order to compare the results of the new Q-learning algorithm and the classic Q-learning algorithm. It is shown that the new Q-learning algorithm accurately acquires the desired outcome in shorter iteration cycles, while having the same time and space complexity as the classic Q-learning algorithm.

Introduction

Reinforcement learning is an algorithm that requires only a scalar reinforcement signal as performance feedback from the environment. One of the difficult problems that reinforcement learning faces is known as the temporal credit assignment problem (Sutton, 1984). For example, suppose a system consists of a large number of discrete states. When we update the Q-value of a certain state, the new Q-learning algorithm accurately acquires the desired outcome in shorter iteration cycles, while having the same time and space complexity as the classic Q-learning algorithm.

model of the environment. However, it has been found that the Q-learning algorithm is not suitable for a system that has a large number of states (i.e., a game), as it takes too much time for a machine to learn the desired outcome. (Sutton, 1990)

In this paper, we focus on developing a new Q-learning algorithm that learns a system that consists of a large number of states successfully in shorter iterations. This research is significant because it suggests a new machine learning algorithm that is simple and intuitive, yet powerful enough to be implemented into a complex game.

Algorithm Design

Classic Q-learning

In the simplest form, one-step Q-learning, is defined by:

\[ Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)] \]  

(1)

Where \( r_{t+1} \) is the reward observed from \( s_{t+1} \), and \( \alpha (0 < \alpha \leq 1) \) is the learning rate (may be the same for all pairs). The discount factor \( \gamma \) is such that \( 0 < \gamma < 1 \). In this case, the learned action-value function, \( Q \) directly approximates \( Q^* \), the optimal action-value function, independent of the policy being followed.

Note that only the Q-values of one state can be updated at each iteration. Therefore, if a system consists of a large number of states, it would take many iterations to master the desired behavior. As such, the classic Q-learning algorithm is not efficient for a complex system that has many state-action pairs, such as a game.

New Q-learning

It is shown that the classic Q-learning algorithm does not function successfully if a system is too complex. When the problem space is too large to explore completely, a learning agent must have the ability to guess about new situations based on prior experiences from similar situations.

There are some problems where we can use previous knowledge, gained through prior experiences, to gain insight about a new situation. A system in which neighboring states possess similar characteristics is one such problem.

Suppose that a system consists of a large number of discretized states. When we update the Q-value of a certain
state, we can also give the similar Q-values to the neighboring states, assuming they would have similar characteristics. Clearly, this general pattern decays eventually - that is, there is a threshold that a state cannot be considered as a neighbor state of the target state anymore. We can model this by updating the Q-values of the neighboring states to a linearly decreasing level. If we decide to update the \( n \) number of adjacent states, the target state would get the original new Q-value, and all the \( n \) neighboring states would have smaller Q-values than the Q-value that the target state gets, under the rule that the farther states get the linearly lower values.

This linear method may not be the most accurate way of guessing the Q-values of the neighboring states, but it is very simple and intuitive, and it does have the concept of a decay and threshold. The following lines of C# codes are the excerpts that describe this process:

```csharp
// Q : (1 - alpha) * Q + alpha * (reward + gamma * maxQ)
// n : update +- n neighboring states
// Assume that a system has a state level from 0 to 100.
public void updateQValue(double Q, Action action)
{
    for (int i = Math.Max(0, this.getState() - n);
        i <= Math.Min(100, this.getState() + n); i++)
    {
        double dist = Math.Abs(i - this.getState());
        double newQ = this.qvalues.getQValue(i, action);

        newQ = (Q - newQ) * (n - dist) / n + newQ;
        this.qvalues.storeQValue(i, action, newQ);
    }
}
```

Through this process, we can have information on a large number of states within a smaller number of iterations. Figure 1 shows the detailed process of the algorithm.

The disadvantage of the algorithm is that it would learn the system accurately only if the assumption that the neighboring states have the similar characteristics is valid. Thus, the use of this algorithm is limited to certain systems. However, for a certain system in which the neighboring states have similar desired outcomes, this improved Q-learning algorithm will be very effective.

### Implementation

The Q-learning algorithm is analogous to training an animal to behave in a certain way by giving some type of reward or consequence. Over time the animal learns to perform in the desired way faster and more efficiently.

This interesting concept can be implemented in various types of games, in that a non-player character (NPC) of a game can be modeled after an animal. A game that implements our algorithm should consist of a number of discrete states, where adjacent states have similar characteristics. As such, a resource management game serves as an example.

### Resource Management Game

Currently in the market, there are a number of resource management games that have been very popular and successful (i.e. FarmVille, The Sims, etc.). A resource management game usually involves a user who manually controls the decision to produce or consume a certain resource in order to attain a higher goal. In order to succeed in the game, a player makes decisions based on a ‘general’ strategy regarding the amount and kinds of resources available to a user. From the perspective of the Q-learning algorithm, we can regard the amount of each resource as a discretized state. Based on the assumption that a user would want to make a similar decision given a similar situation (i.e. if a user is given 100% of the resource, they would make the a similar decision if they had 99% of the resource), we can implement our improved Q-learning algorithm to build an artificially intelligent system. By assigning similar Q-values to the adjacent states of the target state, we can train NPCs in shorter iterations to learn the desired behavior.
Example: AResome

Using the concept above, we’ve created a user-interactive 3D game in an Augmented Reality called ‘AResome.’ ‘AResome’ consists of a number of NPCs, that consume or produce available resources. A user gives either a positive or negative reward to every action that each NPC takes, which teaches NPCs to behave in a manner that the user wanted. When using our improved Q-learning algorithm the learning period is comparatively short regardless of the number of resources and discrete levels - that is, if a user carefully gives feedback to the NPCs, they will learn to behave in a ‘sustainable’ manner, based on previous feedback from the user. As seen in the case of ‘AResome’, we can create a very interactive and educational game by implementing the improved Q-learning algorithm.

Results and Analysis

In order to evaluate the success of the new Q-learning algorithm, a test bed was designed and run several times to collect data on the system. A learning algorithm is deemed successful (1) if it learns the desired outcome accurately in a smaller number of iteration cycles, while (2) the algorithm maintains a decent time and space complexity.

Learning Period

The purpose of the learning algorithm is to make a system to learn a desired outcome given a situation. In the Q-learning algorithm, a user can ‘induce’ the system to behave in a certain way by giving a positive reward to the desired outcome, and a negative reward to the undesirable outcome. Through repetitive feedback to the system, a user can eventually train the system to learn the desired behavior and maintain the favorable state. The most important property of the algorithm that is being tested is the speed at which the system can learn the behavior - a successful learning algorithm would master the desired behavior within a short amount of iterations, while an inefficient learning algorithm would take a longer time to learn the behavior. Based on this logic, a test bed cumulatively measures the difference between the value of the desired state and that of the actual state during the given number of iterations. That is, we define the average performance as:

$$p_{avg} = \frac{1}{n} \sum_{i=1}^{n} |S_{\text{desired}} - S_i|$$  \hspace{1cm} (2)

Figure 3 shows the average result of a system in which the total number of states was 20, the desired outcome (the level of the state) was 60, alpha was 0.4, gamma was 0.5, and exploration constant was 0.2.

Notes and Analysis

The figure below shows the average result after running the test bed 100 times for each number of iterations, where the total number of states was 200, the desired outcome (the level of the state) was 60, alpha was 0.4, and exploration constant was 0.2.

Note that a smaller $p_{avg}$ indicates that an agent learns the desired outcome faster. As it can be seen in the graph, the classic Q-learning algorithm takes more iterations to master a the correct behavior for each state. The improved Q-learning algorithm, however, learns the desired behavior much faster because it assumes that the adjacent states would have similar Q-values. Therefore, we can conclude that the improved Q-learning algorithm performs better than the classic Q-learning algorithm when the system is complex, or the number of states in a system is large.

Figure 3 shows the average result of a system in which the total number of states is 20, the desired outcome (the level of the state) was 6, alpha was 0.4, gamma was 0.5, and exploration constant was 0.2. Note that the total number of states was only 10% of the previous case.
Figure 3: The result of a test bed run from a system with 20 states, which shows the average performance values after \(n\) actions. \(n\), or the number of iterations (actions) is along the x-axis, while the average performance value at each \(n\) is along the y-axis. Note that the new Q-learning algorithm (colored blue) and the classic Q-learning (colored red) take roughly the same number of iterations to reach the lowest performance value.

Figure 3 shows that the improved Q-learning algorithm and the classic Q-learning algorithm have roughly the same performance when the problem space is not large enough.

In summary, it can be concluded that the improved Q-learning algorithm can perform more successfully than the classic Q-learning if the system consists of a large number of states.

**Time and Space Complexity**

The improved Q-learning algorithm follows most of the process that the classic Q-learning algorithm goes through. The only difference comes when it updates the Q-values after a reward is given; instead of updating the Q-values of only the target state, the improved Q-learning updates those of the neighboring states as well. Since it has only finite more number of computations (i.e. if it updates the Q-values of the 10 neighboring states, it takes only 10 times more time than the classic Q-learning algorithm), it does not affect the overall time complexity of the algorithm. Therefore, the Big O notation of the improved Q-learning algorithm remains the same as that of the classic Q-learning algorithm.

In addition, the improved Q-learning algorithms and the classic Q-learning algorithms have the same space complexity, as we are not creating any more data structure.

Therefore, we can conclude that the improved Q-learning algorithm and the classic Q-learning algorithm have the same time and space complexity.

**Conclusion**

We proposed an improved Q-learning algorithm that learns the desired behavior more quickly and efficiently than a classic Q-learning algorithm. The new Q-learning algorithm can only work under the assumption that the neighboring states would have similar Q-values. Therefore, this algorithm can be implemented in only a specific type of system, one in which the adjacent states share similar characteristics. A resource management game is an example of such a system in that a user would make a similar decision given a similar amount of resources. A user-interactive 3D game ‘AResome’ was newly designed, which showed that the improved Q-learning algorithm can successfully be implemented into a complex game.

In addition, a test bed was designed in order to compare the results of the new Q-learning algorithm and the classic Q-learning algorithm. When tested on a system that had only 20 states, the results showed that the classic Q-learning algorithm and the improved Q-learning algorithm had similar performances. However, when they were tested with a system that had 200 states, the improved Q-learning algorithm learned the desired states much faster than the classic Q-learning algorithm. Also, it was shown that the improved Q-learning algorithm has the same Big O notation as the classic Q-learning algorithm. Therefore, we can conclude that the new Q-learning algorithm accurately acquires the desired outcome in shorter iteration cycles, while having the same time and space complexity as the classic Q-learning algorithm.

Our future plans include the investigation of a more sophisticated mathematical model to update the Q-values of the neighboring states. In this paper, we used a model that updates the Q-values of the neighboring states to a linearly decreasing level. We expect that the performance of the algorithm can be further enhanced by updating the Q-values in an adaptive manner.

We also plan to further develop ‘AResome,’ which is the game that implements our improved Q-learning. This will show that our algorithm can successfully be implemented in a very complex system.

**Reference**


Acknowledgments

We would like to acknowledge the contributions made by Professor Eni Mustafaraj and the students in the Artificial Intelligence class taught at Wellesley College during the fall 2010.