Artificial Intelligence using the QLearning Algorithm

J O H N   F O R S B E R G
a n d   L U K A S   M A T T S S O N

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2010

Artificial Intelligence using the QLearning Algorithm

J O H N   F O R S B E R G
a n d   L U K A S   M A T T S S O N

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Johan Boye
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
forsberg_john_OCH_mattsson_lukas_K10006.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Preface

The project consisted of four parts:

- planning
- programming
- execution
- report writing

Some of them were done simultaneously, such as planning and programming. The planning, execution and report writing parts of the project were done by both members of the group and the programming was done completely by John. The execution part of the project was not much work but is still mentioned. While John was concentrating on the programming part Lukas was focused on structuring and writing the report. Because only John could program proficiently in the language C++ and Lukas was better at writing texts the workload was divided this way.

Abstract

This report is about the AI Learning algorithm QLearning. More specifically it is about how the algorithm performs when given a limited amount of time to learn. In this report several different configurations of the QLearning algorithm as well as other more simple AIs will be competing against each other to give a result as to which configuration is the most versatile.

Referat

Artificiell Intelligens baserad på QLearning algoritmen

Denna rapport handlar om AI-inlärningsalgoritmen QLearning, eller närmare bestämt om hur den presterar givet en begränsad inlärningsperiod. I denna rapport kommer flera olika konfigurationer av algoritmen samt andra simplare AI:s tävla mot varandra för att visa vilken konfiguration som är den bästa.

Contents

Chapter 1

Introduction

In the field of implementing game AIs there have been many different approaches during the years. Everything from the simplest models that are hard-coded to do certain things at certain times such as the mushrooms in Super Mario which walk to the left if Mario is to the left and right if he is to the right, or the more advanced models such as those in the game "F.E.A.R." which perform complex tasks such as communicating with each other and giving covering fire to allow their teammates to spread out to flank you[5]. Most game AIs do however have a fatal flaw: They do not learn from their mistakes! They are simply instructed by the programmer to behave a certain way in a certain situation, and if the programmer missed one situation the AI will not have a proper response for it. This is where machine learning comes into the picture.

1.1 Background

Machine learning is the subject of teaching computer programs how to learn for themselves, and has been around since the 1960's[4]. With a machine learning algorithm the AI would gain what its programmer would not be able to give it in advance: the ability to react to unforseen situations.

When talking about learning algorithms there are different categories[2]:

- Supervised Learning
- Unsupervised Learning
- Semi-supervised learning
- Reinforcement Learning
- Transduction
- Learning to learn

This report covers the Reinforcement Learning algorithm QLearning[3]. For information about the other categories and examples of algorithms for the respective categories, see Wikipedia on machine learning[2].
A Reinforcement Learning algorithm learns from past experiences. It is given a series of actions to choose from and after every choice it has made the environment can give it a reward or a punishment, which is a negative reward. This can be set up in many different manners but the common theme is that the algorithm will strive towards the option that has the highest reward associated with it.

1.2 Problem Statement

The question that we are trying to answer is: How good is the QLearning algorithm? Following are the main questions we will be covering:

- **How long does it take to learn?**
  How much time does the algorithm need before it can stand on its own and play against an arbitrary opponent without losing all the time?
- **How smart is it?**
  Is it able to predict the opponents tactics and formulate a counter-strategy or will it just keep going on with its own strategy oblivious to its opponent? What are its strengths?
- **Does it have weaknesses?**
  Is it vulnerable to certain strategies or will it perhaps adapt to new strategies very slowly?

When discussing these things we came across another question that we decided to include in our project in order to better understand the results and put them in perspective: Can the QLearning algorithm be improved?
We want to see if it is possible to improve the learning speed of the original QLearning algorithm by using some alternative means of decision making until it has gathered some experience. Thus we also have the following item on the question-list:

- **Can the learning rate be improved?**
  By allowing the algorithm to do random actions until it has gathered some experience, will it learn what to do faster?

1.3 Definitions

Action	An action performed by an agent alters its current state.
Agent	A program entity that uses an algorithm to simulate AI behavior.
AI	Artificial Intelligence. An algorithm with intelligent properties.
Brain	The brain consists of all the learned knowledge for an agent.
State	A state is an exact representation of the environment the agent operates on.
Tile	A tile is a location on the game board. It can either be empty or occupied by one piece from either player.

Chapter 2

Method and Results

2.1 Method

In this report different types of agents will be used in both the learning and evaluation process to have something to compare the modified QLearning algorithm to:

- A Naive AI.
- A Classic QLearning AI.
- A Modified QLearning AI.

A program entity that uses an algorithm to simulate AI behavior is called an *agent*. The agent is at every given time in a *state* and for each state there is a number of possible actions. Every state and possible action for that state has a reward assigned to it. In our implementation a state is a description of how the game board currently looks and an action means placing another marker onto the board, altering its state. All of the three following are agents although they have different behavior.

2.1.1 The Naive AI

The Naive AI is a non-learning algorithm which follows a simple set of rules when deciding what actions to take. It is therefore very predictable.
The Naive AI begins by placing a marker at one of a predefined set of tiles at random, and in each successive move it will place a marker at an empty tile adjacent to the previous one. This will give simulate a player placing markers in a row in one direction until it hits an obstacle, either an occupied tile or the edge of the game board; and then take a turn to begin making a row in another direction.

2.1.2 Classic QLearning AI

The agent called Classic QLearning AI will follow the QLearning algorithm except for some smaller changes, these changes are stated in Problem Execution, section 2.2.

The QLearning algorithm is presented in equation 2.1. The algorithm works by retrieving previous learned values by mapping a state with an action. $Q\left(s_t, a_t\right)$ is the mapping of state s and a at time t which yields the reward for using that action at that state. To make the implementation of the algorithm easier, the *learning rate* $a_t\left(s_t, a_t\right)$ function is removed and set to a constant of 100%. What remained was the γ which is called the discount factor that specifies how important the next action will be, it is a number between zero and one. The most essential part of QLearning is as previous stated the retrieval of the reward. It has two appearances in the algorithm, the first being $Q\left(s_t, a_t\right)$ which retrieves the reward for the current state s_t and the action a_t. The second part as seen in equation 2.2 uses the best action after the first action has been performed and therefore use the updated state s_{t+1}.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a_t(s_t, a_t) \times [r_{t+1} + \gamma \times max_a\{Q(s_{t+1}, a_t)\}] \tag{2.1}$$

$$max_a\{Q(s_{t+1}, a_t)\} \tag{2.2}$$

2.1.3 Modified QLearning AI

The Modified QLearning AI is very similar to the Classic QLearning, the only difference being that the Modified version will pick actions at random in the beginning. This difference will significantly alter the way that the AI looks for mistakes after losing a round. Instead of testing tiles sequentially in order to block the opponent from winning it will try placing them at random, hopefully finding the correct move faster.

Before the modified agent performs an action a check is made to determine if it should be a randomly chosen action or if it should use its experience to choose the best action. The chance for a random chosen action decreases at a rate linear to the knowledge in the brain. A factor that determines how much knowledge it should have before it stops is implemented and in the pseudo code sample below it is called *f*. At the first ten matches the agent performs nearly 100% random actions but at a few hundred matches later there is no random actions.

```
$R \leftarrow randomNr[1, 128]$
$S \leftarrow numberOf(StatesLearned)$
if $R - S * f > 0$ then
  {perform a random action.}
else
  {follow the normal QLearning algorithm.}
end if
```

2.1.4 The testing platform

In order to answer the questions in the problem statement the algorithms will be put in a relatively simple environment which is both easy to visualise and to judge: A board game. The board game will be a somewhat modified version of the classic game three-in-a-row. In the modified version (from here on out called five-in-a-row) two players take turns placing their markers on a game board consisting of 9x9 tiles, and the first player to place five of his markers in a row either horizontally, vertically or diagonally wins the game. The evaluation will consist of two phases:

Training

In the first phase one fresh copy of every type of agent will be assigned a sparring partner. They will then complete 400 game rounds against each other.

The matchups for the training will be as follows:

- Classic QLearning vs Naive AI
- Modified QLearning vs Naive AI
- Classic QLearning vs Classic QLearning
- Modified QLearning vs Modified QLearning

The learning experiences of the learning AIs from this training will then be saved and copied for the second phase. To keep track of all the different AI experiences they must be named uniquely. Their names will thus be either "Classic" or "Learning" followed by the first letter of their opponents name, either "N" for Naive, "C" for Classic or "M" for Modified.

Thus, the AIs available after the training are:

- ClassicN
- ModifiedN
- ClassicC
- ModifiedM

Competition

In the second phase the differently taught agents will play against each other in all possible matchups for 200 rounds. To ensure that no single AI gets to play several competition games and thus gain an advantage by having more experience, copies of the different AIs will be made; one for each matchup. The AI with the most games won will be considered the winner.

The matchups for the competition will be as follows:

- ClassicC vs ModifiedN
- ClassicC vs ModifiedM
- ClassicN vs ModifiedN
- ClassicN vs ModifiedM

2.2 Problem Execution

The code structure was planned in conjunction with researching implementations of the QLearning algorithm and it was finally decided to implement it in the language C++. A website with the QLearning algorithm in C++ was used to get a good idea on how to begin[1]. The programming was by far the most time consuming part of the problem execution, both due to the nature of the task and because only one of our two project members were comfortably familiar with the language.

First the game was implemented. It was structured to allow both human and computer players to make modifications to the board according to the same set of rules. A logging function was also added to the program for readability of the data.

The Naive AI was created very quickly due to its simple nature. The only thing that was added to its functionality in addition to that specified in 2.1 was the ability to alter its starting coordinates using the configuration file mentioned in the next paragraph. The implementation of the QLearning algorithm was straightforward except for one thing: As seen in equation 2.1, the algorithm takes the currently available actions as well as the potentially available actions for the next round into consideration when deciding what the next action should be, the problem is that between these two actions the other player is taking an action. To overcome this problem the implementation finds the best "next action" by looking through its currently available actions and for every one of those find the opponents best available actions and for every one of *those* search for the next best action. This is a time consuming yet necessary step to adapt the algorithm to the game.

After all the code was written different agents could be configured by editing an external configuration file for each matchup as specified in 2.1. The agents would be created using the configuration file and if the user wanted them to use any previously learned experience he had to specify the name of the file where that knowledge had been saved during the configuration. After the specified number of games had been played the agents would add their new knowledge to that file. This was relatively easy to do but the games were processor intense. Each game could take up to several seconds which meant that the training for every agent took up to an hour on the commercial CPU's used.

X	O	X	O	O				
O	X	O	X	X				
X	O	X	O	O				
O	X	O	X	X				
X	O	X	O	O				
O	X	O	X					
X	O	X	O					
O	X	O	X					
X	O	X	X					

Figure 2.1. The checker board often created by two agents.

2.3 Results

The following section will cover the results for the two parts of the execution, first is the training and following that is the results of the tournament.

Training

The agents training against the Naive AI learned to beat it surprisingly fast and after just a few matches they started to win every time. Because if this the number of alternative starting tiles for the naive AI was increased from one to seven and the training was redone. The agents training against themselves showed a tendency to create a checker board pattern, see figure 2.1.

Tournament

The tournaments was a huge disappointment due to the fact that a bug in the program caused the agents to not learn anything new. Because of this most of the matches that were played presented only two patterns. One pattern when the winning agent started and one for when the opponent started. Without the bug the agents would have learned from the previous match and tried a different action before their opponent won. However this was not the case, and the agent kept losing the same way. The result of this was that the agent using the classic algorithm won all but twelve of its matches.

Chapter 3

Discussion

3.1 Problems

This project consisted of a lot of programing which proved to be very rough due to the choice of language. Also, the fact that all the code was implemented from scratch ment longer implementation time. A better approach would have been to use an already implemented QLearning algorithm, however the implementation of the code was a very good way to understand how QLearning works and the knowledge gained could prove valuable in the future.

Training

During the training of the agents we hoped to see a difference between the modified and the classic QLearning algorithm, but the modification of the algorithm was pointless. The modified version just tried to place markers at random tiles for a couple of matches losing every time, after which it started playing identically to the agent following the classic QLearning algorithm. The only difference between the classic and modified agent was therefore in the end that the modified was missing valuable experience.

Tournament

As previously mentioned there was an unidentified bug during the competitions between the agents, this bug was not fixed due to the late discovery of it. Luckily this was not a huge problem as the tournament was only going to be used to determine which of the agents was the best and since the bug affected all learning agents in the same way the playing field was even.

3.2 Conclusion

Improving the algorithm

As the tournament results showed the classic QLearning algorithm is superior to one modified to take random actions in the beginning. We conclude that the reason for this is because the random actions were used in an ineffective manner. Random actions should instead have been taken when there are no known good actions to take in the current state. Using this as a criteria would have enabled the algorithm to take random actions long into its learning process but only when needed. The random actions would eventually seize, but only when the algorithm no longer needs them.

The reason we thought that adding an element of randomness would help the agent is because instead of iterating through all the possible actions it would have a statistically advantageous chance of finding an effective counter-move because it would not be as predictable and thus find ways to block its opponent faster.

Learning speed

Like any kind of reinforcement learning algorithm QLearning takes a very long time to learn. This is because it must learn to counter every strategy individually, it cannot see patterns or similarities in strategies and therefore anticipate them. In our implementation the learning was generally even slower because we lacked efficient training data. Having efficient training data which simulated a very skilled opponent would encourage the AI to learn faster by not wasting any time playing matches which are very improbable of occurring or identical to previous ones. The naive AI proved to be a more effective teacher than letting the AIs play against themselves because it was a better human analog than the untrained learning algorithms. In conclusion the QLearning AI takes a long time to learn but it can be sped up using training data to simulate the situations it will be facing once its training period has ended.

Strengths and weaknesses

The QLearning algorithm is not very smart. It can not predict opponent strategies and it only knows what it has previously seen. It is unable to recognize patterns and it takes a long time to learn why things happened because it has to iterate through all possible reasons one game at a time. This means that it is very vulnerable to a witty opponent that tricks the AI by using different starting moves and then the same end strategy. It is however a decent algorithm to use in smaller games with a finite number of states because they can be iterated over and learned in a reasonable amount of time.

We can conclude that our experimental modification of the QLearning algorithm was ineffective, but it was not entirely futile for we have, just like the agents we created, gathered experience.

Bibliography

[1] QLearning in C++. http://www.compapp.dcu.ie/~humphrys/Notes/RL/Code/code.q.html. *Last Accessed 10/03/2010.*

[2] Wikipedia on Machine Learning. http://en.wikipedia.org/wiki/Machine_learning. *Last Accessed: 28/04/2010.*

[3] Wikipedia on QLearning. http://en.wikipedia.org/wiki/Q-learning. *Last Accessed: 28/04/2010.*

[4] Samuel A.L. *Some Studies in Machine Learning Using the Game of Checkers.* Feigenbaum and Feldman, 1963.

[5] Jeff Orkin. Three States and a Plan: The A.I. of F.E.A.R. http://web.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf. *Last Accessed: 02/05/2010.*