Matchbox Educable Noughts And Crosses Engine In Empirical Modelling

1362452

Abstract

Matchbox Educable Noughts And Crosses Engine (MENACE) is a machine proposed by Professor Donald Michie in 1960s. This machine is used to learn the Noughts and Crosses game automatically and will become smarter via playing against human. In this paper, this MENACE machine will be applied in Empirical Modelling. The model will use the same method to create hundreds of boxes, and use beads to represent probabilities in making moves. The dependency between the chances in move making observables and the beads in boxes can be clearly seen through the game section and the relevant numbers. It is more understandable to apply MENACE in EM, and easier for others to improve it.

1 Introduction

In this paper, Empirical Modelling is used to build a self-learning machine based on Matchbox Educable Noughts And Crosses Engine (MENACE). The original method of MENACE was proposed by Donald Michie. He was a British researcher in artificial intelligence. [1] During the war, he was working with Jack Good and Alan Turing in Bletchley Park on machines in 1960s. In that time, he got the idea of MENACE to build a self-learning machine to learn the noughts and crosses game. And maybe this method could realize in other similar ways in Artificial Intelligence. [2]

EM is a sense-making activity. The principle of observables, dependencies and agents makes MENACE more sensible in coding or method learning. The different boxes and beads appeared in EM can be easily introduced to other people including programmers. This is the benefit brought by EM in making this model.

This is the first part which introduced the whole idea of MENACE in EM. In the following second part, the basic theory of MENACE will be presented as well as the principle of the game in case of some readers may not know familiar with it. The third part is the applying method. Through this part, readers can understand the function of each part and the total work consists of them. The fourth part is the analyzation of EM based on my own model building experience. The last part will be a short conclusion of this paper together with some prediction of future work about this model.

2 MENACE and the Game

In this part, some background knowledge will be presented, although most of the readers know the principles of this small game. And the method of MENACE will also be introduced.

2.1 Noughts And Crosses

Noughts and Crosses game (or Tic-tac-toe) is a paper-and-pencil game for two players. The players need to mark empty spaces within a 3×3 grid by turns. It is believed to be first appeared in the Roman Empire. [3] The winner is defined as the first one who placed three marks in a row, column or diagonal.

However, most of the time this game will end in a draw. This is because two smart players can easily block each other's way in marking the last square. Although it is not suitable for adults to play, it is ideal in analysing Artificial Intelligence or for learning coding language. [4]

2.2 Principles of MENACE

MENACE, as in its name, is built of different

matchboxes. Each of them represents one game state. If it starts from the AI player, there should be 304 matchboxes. After human player's turn, this machine will check the corresponding matchbox to get the next square for marking.

In each matchbox, there are some different colour beads. Each colour represents one certain square in the 3×3 grid. For the reason that some of the squares are taken by marks, it is necessary to remove the beads of the surplus colours from the matchboxes.

During a game, if starting from the AI player, the machine will find the first one matchbox. Then it takes a random bead from the box to get the random colour. After record the colour of that bead, the bead will be put back into the matchbox. Since each colour represents one certain square, the machine get the square to start this game. If it starts from the human player, the machine will check all the boxes to find the corresponding matchbox to get the next move after the human player placed mark.

When the game comes to an end, the machine will check the result. If the human player won, it will remove each of the used beads from relevant matchboxes. However, if there is only one bead for that colour left, the bead will not be removed in order to make the game continue smoothly.

If the AI player won, the machine will increase each of the used colour beads by one. This will increase the chance for the AI player making the same winning move.

If this game came to a draw, the machine will change nothing but restart a new game.

After thousands of games, the beads in each matchbox will reach quite a large number. This means in certain situation, the AI player has a strong probability to play smart moves.[2]

3 Applying in EM

This part is the applying method of MENACE.

3.1 Platform of EM

For this model, 'tkeden' is used as a platform because there have been several noughts and crosses game models which are convenient to use as a groundwork. EDEN is the Engine for DEfinitive Notations. It is the primary software tool used in the Empirical Modelling for building models. With the variety of definitive notations, we can build models which are understandable and sensible.[5]

Using this in applying MENACE will benefit us in building matchboxes and get random beads. They are observables, and the dependencies between them can be shown to us in eden. It can make sense in EM to build such boxes.[5]

3.2 Matchboxes

The matchboxes in this model will be replaced by different arrays. In order to distinguish them, the function is used in the original codes: box number=s1+1+(s2+1)*3+(s3+1)*9+(s4+1)*27+(s5+1)*81+(s6+1)*243+(s7+1)*729+(s8+1)*2187+(s9+1)*6561. From s1 to s9, they only have -1 or 0 or 1 three different values. Therefore each of them plus 1 to get the positive values and times 3 to make difference. Although there would be thousands of arrays which are not used, it is the only way to create the different matchboxes we need in EM.

For example, if the game start from the AI player. Then the first game state is from s1 to s9, they all 0s. Then we can use this function to get the box number=0+1+(0+1)*3+(0+1)*9+(0+1)*27+(0+1)*81+(0+1)*243+(0+1)*729+(0+1)*2187+(0+1)*6561. And we then can get the result which is 9841. Therefore, the first state of this game is saved in the box 9841.

3.3 Beads

In EM, the beads are replaced by numbers. In order to distinguish them, the arrays are named as boxplace_1, boxplace_2, until boxplace_9 to represent each square. Then combine 3.2, we can get the first square in the first box will be boxplace_1[9841] which means there is no square has been marked. And the number inside of boxplace_1[9841] is the number of that beads.

During a game, in order to get the random bead, a method similar to rain drops is used in this program. As in Figure 1, there are nine blocks waiting for the rain.

Figure 1.The rain drop blocks

When the rain begins, the drops will drop into the nine blocks randomly. Then the first drop will decide which block is the random result.

In my model, each block means each square in the grid. And the numbers of the beads represent the length of the blocks. For instance, the number in boxplace_1[9841] is 9, then the first block is with the length 9. If the rest of the blocks remain small lengths, then it would be like Figure 2.

Figure 2. Boxplace_1 is larger than others

In this case, the probability of the rain dropping into the first block is increased, meaning the AI player has a high probability to play a winning move.

To realize it in math, the function '%' is used to get the remainder of the random number from rand(). The divisor is the total length of the blocks which are the summary of the nine numbers form nine squares. Once got the result, it will be compared with eight numbers, which are bead1, bead2 … bead8. The bead1 is just the number in first square, and bead2 is the total length of the two numbers in first two squares. Then bead3 is to add up the numbers in the first three squares, and repeat until got eight of them. If the remainder is larger than bead4 but smaller than bead5, it means the rain drop in block 5, and the next move is to place mark in square 5.

To explain this more clearly, there gives an example. Let's still use the box 9841 which is the first box. In boxplace_1 of it, the number is 9. Assume rest of the boxplaces are all 1. Then we can get bead1 = 9, bead 2 = 10, bead3 = 11, bead4 = 12, bead5 = 13, bead6 = 14, bead 7 = 15, bead8 = 16, the total summary of them is 17. Once we got a random number which is 789 in this example, we use 789%17, and get the result 7. It is smaller than bead1. Then the next move is to mark square 1.

3.4 Process of the game

The whole process of the MENACE machine is shown as Figure 3. It is built of 3 different parts. The first part Initialization is different compared to that when we restart the game.

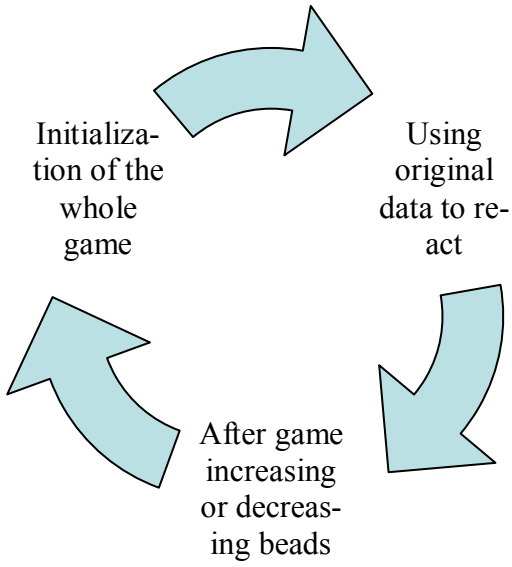

Figure 3. Process of the game

3.4.1 Initialization

At the beginning of a game, the first step is to build thousands of different arrays. Because the max number of box numbers is 19602, we need to create nine arrays each with 20000 values. The codes for this are:

```
%eden
boxplace_1 = [];
for (i=1; i<=20000; i++) {
        boxplace_1 = boxplace_1 // [1];
};
```

Besides, another two small arrays to record the used box numbers and the used beads should also be initialized. There are nine moves and nine squares in total. Therefore both of them are defined as follows:

```
%eden
usedmove = [];
for (i=1; i<=9; i++) {
        usedmove = usedmove // [0];
};
beadsnumber = [];
for (i=1; i<=9; i++) {
        beadsnumber = beadsnumber // [0];
};
```

This is the initialization of the first time running this game. When one game is over, the data in each box will not be replaced by 1 as above. This will gradually increase different probabilities in placing different marks.

3.4.2 Reaction to human players

Once the game started, the game state will be sent to the matching part. When it is AI player's move, the computer will go to the corresponding box to load the information inside. Once got values for bead1 to bead8 and bead9 (which is the total summary of the nine numbers, it is named as bead9 for convenient use), we can use bead9 to divide a random number got from rand(). Then we need to compare the remainder with bead1 to bead8 to get the square number for the next move.

However, in the initialization part, the values in all the boxes are all set as 1s. Therefore, it is necessary to build a new function to check it. I used checksquare() to do this. It will work after got the random bead. If the random bead represented a

marked square, the value of that square in this box will minus 1 to set it as 0. Then repeat the get-random-bead step, and recheck. Each turn we only need check no more than 9 times. Therefore it is better to set it as a loop program.

Finally we can get the right move. In the same time, to explain this method to players, a box of current beads numbers is placed near the playing grid. It is also necessary to record the box number and the bead number for after-game process.

3.4.3 After-game process

In this part, the data will be processed. In the original game, if the player who used Os win, the value of owon will be set as 1. Same to the player using Xs, when he wins, xwon will be set as 1. Therefore, I put the function as follows into the files:

```
if (xwon)
        increasebeads();
if(owon)
        decreasebeads();
```

The functions of increasebeads or decreasebeads are written in AIxox.e and in makemove.e file. Sometimes this game will end in less than 9 moves, let say it 6 moves. In order to avoid using the old data in usedmove or beadnumber, the 'steps' was inserted to control it as:

```
for (i=1;i<=steps;i++)
```

In each turn, the steps will increase by 1.

When all these were done, the game will restart if push the button 'INITIALISE'. But this time, it will not go back to the 3.4.1 part, because we want to keep the data in the thousands of boxes. It will just give 0s to all the squares which will make them back to empty.

3.5 Application of this model

Although all the steps are clearly shown above, it is not that easy to apply it in EM with tkeden. Figure 4 is the screen capture of my model.

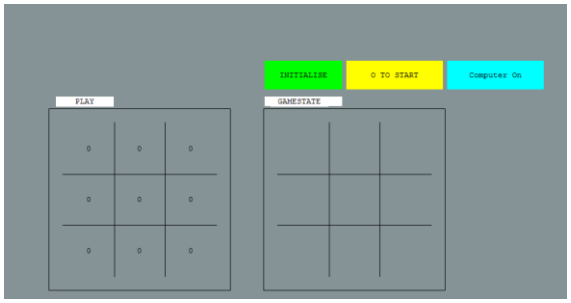

Figure 4. Screen capture of MENACE

It is consisted of two grids: one for play, another one for showing beads.

The following is an example to use this model. In Figure 5, I started the game by clicking the up left corner to mark it as O. And the AI played square 6, which is always the first move when starting this game.

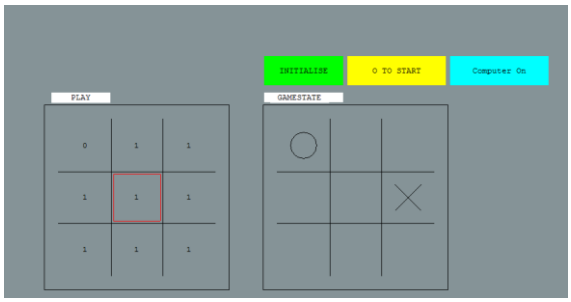

Figure 5. First move of first game

And then try to lose it as in Figure 6.

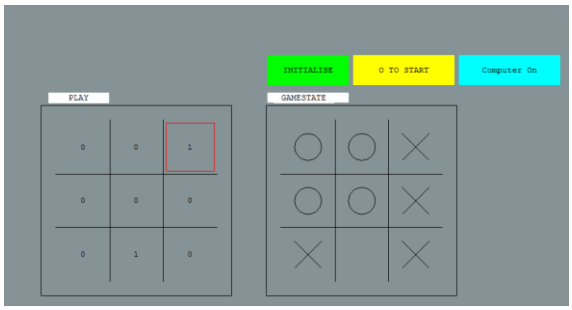

Figure 6.Losing the game

Then we need to restart a new game. Click on the button INITIALISE to get restart. And as in the first game, I clicked on the up left corner of the game to check the beads numbers in the used matchbox. It is shown in Figure 7.

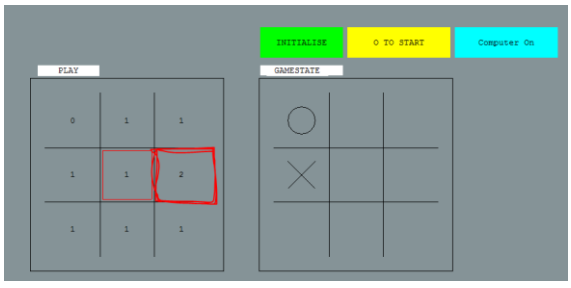

Figure 7. Second game with the same move

As we can see in Figure 7, the number of square 6 in the matchbox is changed to 2. This is because the first time, computer has played square 6 to win the game. Then the beads of square 6 in that matchbox will increase by 1 after click the INITIALISE button.

Although the next move in Figure 7 is not the winning move square 6, which is because the num-

bers are too small, after handers of games, the AI player will become smarter, and this game will come to a draw at the most of the time.

4 Model building Analyse

During the model building project, I found several aspect of EM which are ahead of the traditional coding method, and they are convenient for the programmers.

4.1 Model can be touched

At the beginning of this model-building process, I just read the codes of the original model to change it. Although it is better to understand the whole program through above activity, it is not efficient to do this work, especially in other large projects.

However, in tkeden, it is not necessary to do that. Since the model is based on observables and dependencies, all we need to do is try to change the dependencies (the relationship between bead numbers and next move) and add more observables (the thousands of different boxes).

This part of work can be done by just 'asking' questions about the present observables. It is like touching the model with own hands. And it will benefit the next programmer who will continue the work or extend the current model. This is an efficient way in model-building area.

Although it is beneficial to apply the models in EM, it is a little difficult to do this in tkeden. For example, when using the function rand(), we suppose to get a random number. However, every time when I restart the tkeden program, the first number I got through rand() is always 2078917053, and the second one is 143302914, the third one is 1027100827. And I will get some other problems in using tkeden, too.

It is true that tkeden has errors, but it is not the errors of EM method. Those minor flaws are far outweighed by the advance of EM method. Plus it is already shown advantages in this model building.

4.2 Three foundations of EM

The three foundations of EM is observables, dependencies and agent. In my own model, the observables can be seen through the play grid, which are nine numbers from different boxes. It makes sense to the learners to learn the MENACE method or this model with visible objects.

The dependencies between the matchboxes, the beads and the next move are also can be acquired via tkeden. They connected the current objects as it is in real life. The life-like model is more acceptable in model learning or model building. With the basis of Empirical Modelling, the logical parts can be applied as observables in combining or reconstructing. Just like it is realized in this model making. The original numbers of the squares in play grid is changed by another function to calculate the value of different move. It is used to find the best response for the AI player. The observables are these numbers, and the dependencies are the numbers and the values of current move. I reconstructed the play grid by linking the numbers to the numbers of the beads. It is simply changed as the dependencies changed.

The original agent in this game is AI player and human player. Here, the AI player is not the one I used in my model. This player is playing the game with valuating different squares, and making move to take the one square which is of max value. I changed it into another agent, which works as MENACE.

5 Conclusion and Future work

5.1 Conclusion

In this model building work, I experienced the benefits brought by Empirical Modelling, making it easy to understand and efficient in using previous model to build new ones.

It has shown strong practicality in building models. All I need to do in building the new model is to find the current observables and add my ones, create new dependencies to link them together.

5.2 Future work

At present, this model is not finished yet. Computer palyer should play against itself to get smarter. The first step is try to fix it.

After that, try to reduce the total number of the boxes. Although we are using computers to deal with this program, it is better to reduce the total waiting time of it.

Acknowledgements

I would like to acknowledge Meurig Beynon here. He is so supportive and warmth in helping my work. Without his help, I do not think I can finish this model. And with his guidance, I learnt how to find the important observables in the original model.

References

[1]http://www.aiai.ed.ac.uk/~dm/ Donald Michie home page

[2]http://shorttermmemoryloss.com/menace/

[3]Zaslavsky, Claudia (1982). Tic Tac Toe: And Other Three-In-A Row Games from Ancient Egypt to the Modern Computer. Crowell. ISBN 0-690-04316-3.

[4]Steve Schaefer (January 2002). "MathRec Solutions (Tic-Tac-Toe)". MathRec. Archived from the original on 28 June 2013.

[5]http://www2.warwick.ac.uk/fac/sci/dcs/research/em/software/