# Train Alpha Zero to play Ultimate tic tac toe

Yue Sun

May 2021

**Abstract**

In this project, I train an Alpha Zero that can play Alpha Zero. Different from the original Alpha Zero, the stride of the convolutional layer in Ultimate Tic Tac Toe is 3 instead of 1, because, in the Ultimate Tic Tac Toe game, the is no strong relation among the different mini-board. In the report, I introduce the structure of Alpha Zero and how we achieve it. Then we proof the convergence by experiment.

## 1 Introduction

Ultimate tic-tac-toe is a board game composed of 9 tic-tac-toe game boards. Players take turns to play. in the first turn, the player can play any position on the board. But in the following steps player must the position in the mini-board, which is the same position relative to the entire board, except when the corresponding mini-board is full or not legal to play. see figure 2. It has been proved that the Ultimate tic-tac-toe game is complex and it is hard to solve it with brute force. In my project, I first try to solve it with simple Q learning. However, the convergence rate is so slow that it cannot achieve a good result before the end of the project. Another reason we cannot use Q-learning is the lack of effective game records. Then we switch our agent to an alpha zero-based learner. Such an agent solves the problem I mentioned before. It converges faster and it doesn't need the expert's game records, which is impossible for me because even now I don't know the strategy for this game. We record the loss after training for 46 hours and the loss has a slight downward trend. I also play the agent and I cannot defeat it so far. This is another piece of evidence that I think is a valid AI.

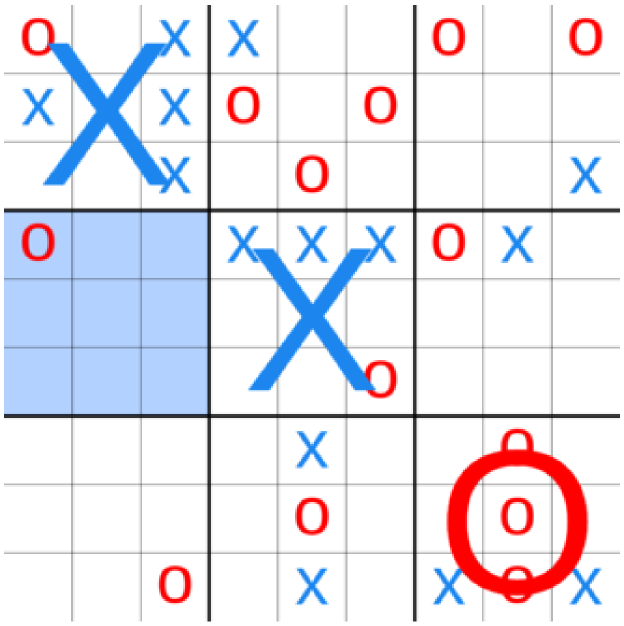


Figure 1: Ultimate tic-tac-toe game

## 2 A Q-learning based agent

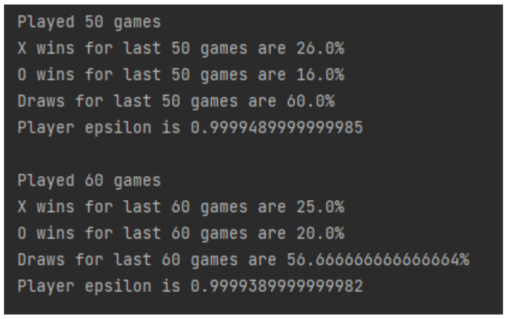
Q learning is a long-established algorithm. A Q-learning agent uses a table to record the value function for every possible state and the action. After playing a game, it updates the Q-table. And after enough iteration, it will converge to the target Q. The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg( \underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \Bigg)}^{\text{temporal difference}}$$

Figure 2: q learning equation

where $r_t$ is the reward received when moving from the state $s_t$ to the state $s_{t+1}$, $\gamma$ is the discount factor, and $\alpha$ is the learning rate $(0 < \alpha \leq 1)$.

In my project, I set $\alpha = 0.9$, $r_t = 0.1$, and $\gamma = 0.97$, and train this agent. The time cost for each iteration is quite fast, because the agent doesn't need a lot of consideration to make a decision. And part of the output is like it is shown in the following figure 3.



```
Played 50 games
X wins for last 50 games are 26.0%
O wins for last 50 games are 16.0%
Draws for last 50 games are 60.0%
Player epsilon is 0.9999489999999985

Played 60 games
X wins for last 60 games are 25.0%
O wins for last 60 games are 20.0%
Draws for last 60 games are 56.666666666666664%
Player epsilon is 0.9999389999999982
```

Figure 3: q learning sample output

From the figure, we see the decrease of $\epsilon$ is quite small. The average is about 1/100000 each iteration. It converges with the same speed, it at least takes about 55 hours to reach the 0.98. I train it for 12 hours and the $\epsilon$ stops at around 0.9915(see figure 3), which is not huge progress compared with a random play. And when I play with it, I find I can beat it easily. Actually, I think it is playing randomly. Thus I think my Q-learning is not a good learner. So I switch to other agents.

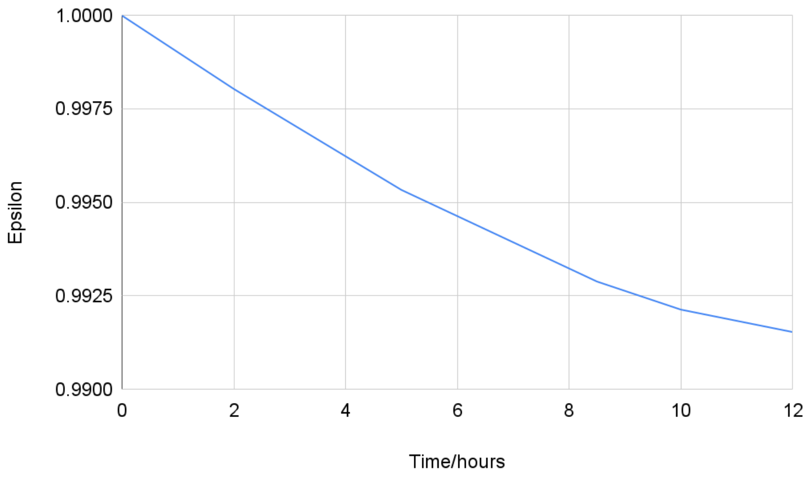


Figure 4: q learning $\epsilon$ changes in 12 hours

## 3 Alpha Zero

Alpha Zero is a powerful agent for learning games. Its original usage is on Go AI and Chess AI. I borrow the alpha zero from the Chess AI. According to the original paper, it has two particularly obvious advantages:

1. Converges faster
2. Doesn't need game record

These are the reasons for borrowing Alpha Zero to our game. These are also the challenges we are facing in playing Ultimate tic-tac-toe games. Alpha Zero consists of two parts, the convolutional neural network, and the Monte Carlo tree search.

### 3.1 Loss function in Alpha Zero

There are 2 parts to the convolutional network loss function, one is cross-entropy, and the other is mean square error. The definition of cross-entropy is:

$$H(p,q) = -\sum_{i=1}^{n} p(x_i) log(q(x_i))$$

Here, p($x_i$) is the target distribution and q($x_i$) is our network. And then after doing the substitution in Alpha Zero's equation, for every turn of the game ($s_{i,i}$, Z) we know the loss is

$$Loss = (Z - V)^2 - [\pi_i]^T log([p_i]^T)$$

Clearly, the loss converges because both the MSE error and cross-entropy are convergent. In the experiment, we can prove the convergence of Loss by observing these two values. This step gives two outputs, policy and value for the policy.

### 3.2 Monte Carlo Tree Search

Not all the states and actions are in the memory of the neural network. If the agent meets a new situation, Monte Carlo Tree the way we solve it. It is like a simulation of human thinking. When a human is playing the game, he or she will try his/her best to list all the "best" choices in his/her mind, and then carefully compare them. That is also what alpha zero does. There is hyperparameter called mcts search times(MST). The MST represents how many games the agent will play in its mind before making the decision. In our experiment, MCT = 600. This is also the reason why alpha zero doesn't need training data to do the behavior clone first. But such a search takes a lot of time in the beginning and consumes a lot of memory.
When we are training the network, after the game is finished, there is a reward

or penalty for every player. Then we can update the tables in the agent. Let Q(s,a) denotes the expected reward for taking action a from state s. Let N(s,a) be the number of times we took action a from state s across simulations. And P(s,)=p(s) is the initial estimate of taking an action from the state s according to the policy returned by the current neural network. From these, we can calculate U(s,a), the upper confidence bound on the Q-values as

$$U(s,a) = Q(s,a) + c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{N(s,a)}$$

In my project, $c_{puct} = 4$.

## 3.3 work flow of Alpha Zero

The structure of the training network is showned below.

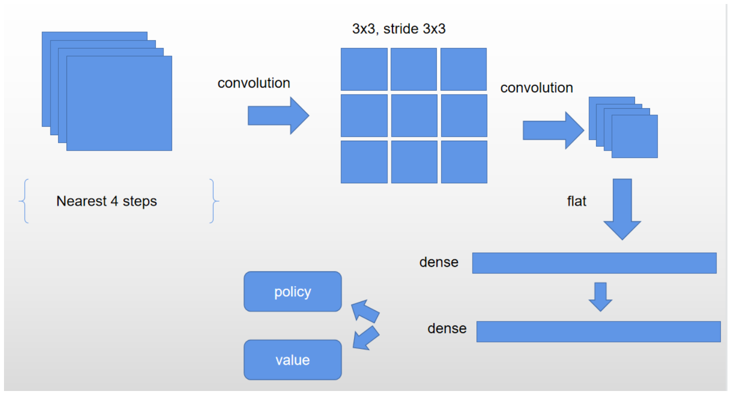


Figure 5: alpha zero neural

The only difference is that we only have 1 convolution layer and the stride is 3x3 rather than 1x1, because the value in the different mini board is not that related. Like it is shown in the following figure. It is a way to overcome the overfitting in this network. Further, I think it is not good to simply add all the board together to get the input tensor since for sure the most closely related is not adjacent in position but adjacent in sequence. That will be in future work.

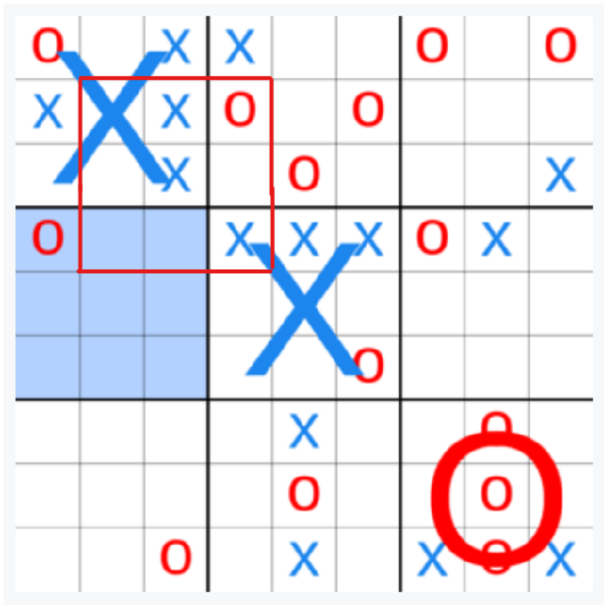


Figure 6: bad convolution example

In the training part, Alpha Zero plays with itself. There is a P record of all the states and actions it meets so far. If the current state is in P, then just return the corresponding value for such a state. Else it will play 600 times with its experience and select the one which wins the most.

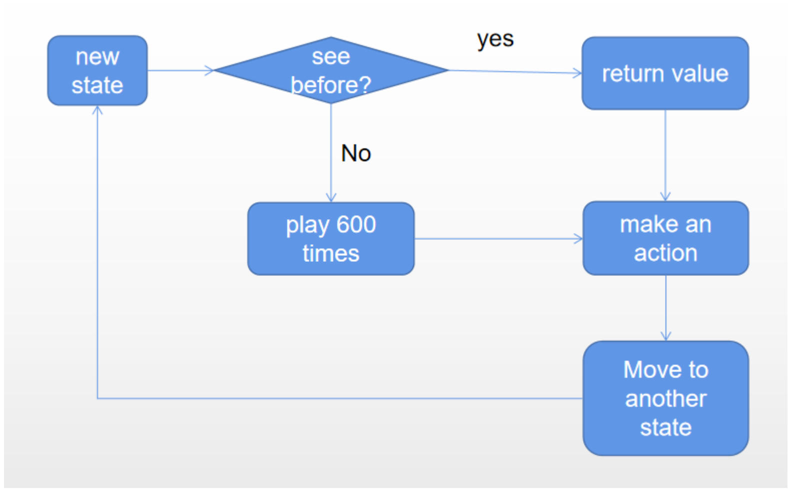


Figure 7: generating play record

At each epoch, the agent plays with itself like this and it will remember its record. Then after each training, we need to check whether the new neural is better than the old version. The new neural will battle with the old version several times. In my project, it is 50. Then if the new neural is better($rate > thresh$), we will update the new neural. Otherwise, we use the old one. In my project, thresh is 0.52.

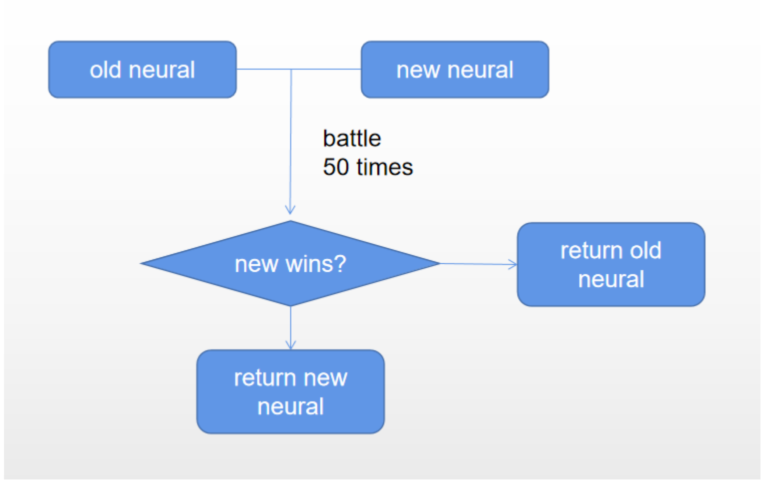


Figure 8: Old VS new

### 3.4 Some notes when training

There are several tips when we train the neural network.

1. Be careful with the zeros on the board. The boards in zeros might be a good notation to empty legal positions. But then it makes the board a sparse matrix, which might make the politics all 0. Thus I don't think the board should not use 0 to denote legal positions. Any value except 0 is ok. In my project, I use 0.1 to denote a legal empty place and -0.1 to denote an illegal empty place.

2. It is better to define the action with an int rather than a tuple. Because there are dense layers in the neural network. Otherwise, it might cause many interesting bugs.

3. The thresh to select a better network should be larger than 0.5. Otherwise, it might fall into a dead loop. Only a better network should be updated.

4. The times for virtual play are important. Arnav Paruthi Suggests 600 is best. Even though it slows the training significantly, it is worth spending so

much time

## 4 Result

I train this Alpha Zero Agent for 30 hours in total before I write this report. The converge is not as fast as it is said in the paper. And the loss is the plot in the following figure. There are some signs of convergence. 30 hours is not enough because it takes about 3 to 5 minutes to finish a game on average and in total it is trained with about 500 games. I think it is not enough. But so far a random player cannot beat it. I cannot beat it either because I am a really bad player in Ultimate tic tac toe. That might be good news for the Alpha Zero.

There are two batches trained separate at each time. Each batch is one total game replay. I plot the separately. From the following plot, we can see the sign of convergence.

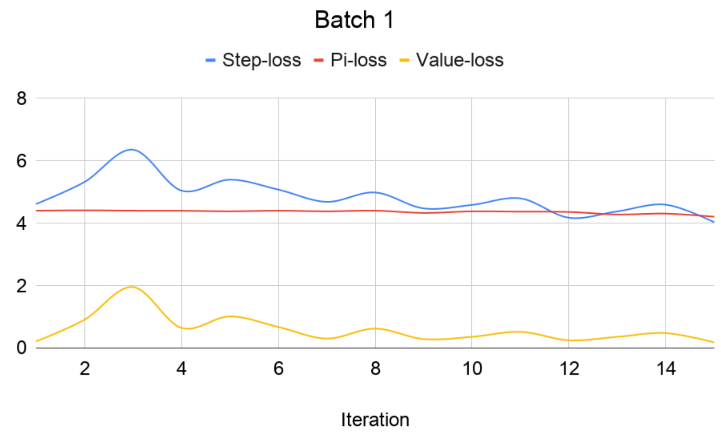


Figure 9: The loss for the first batch at each epoch

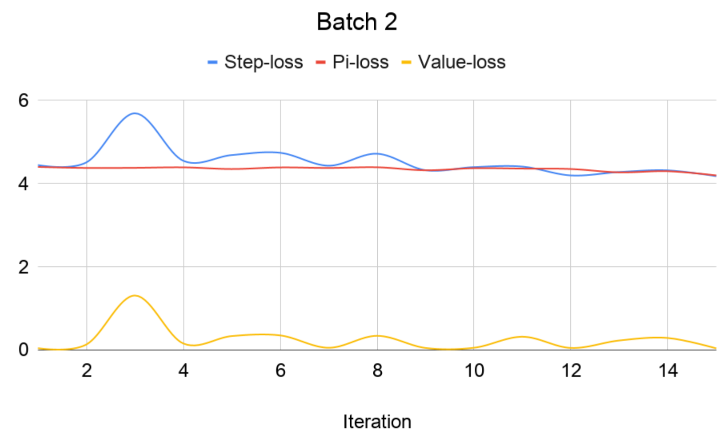


Figure 10: The loss for the second batch at each epoch

## 5 Conclusion

We train an Alpha Zero-based reinforcement learning and summarize the work, difficulty to develop it. I will keep updating it. The project is on my GitHub.

## 6 Reference

1. https://www.youtube.com/watch?v=zHojAp5vkREt=2728s
2. https://web.stanford.edu/ surag/posts/alphazero.html
3. https://www.youtube.com/watch?v=CcwC8tTeQE
4. https://github.com/Arnav235/ultimate tic tac toe alphazero