A Machine Learning Model to play Coganh - Vietnamese chess

Bach Ly and Nguyen Vo
Department of Computer Science and Engineering
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
Email: {bach.ly.ltb1002, nguyen.vo2001}@hcmut.edu.vn

***Abstract*—This assignment is motivated by Alpha Zero, an artificial intelligence computer program with superhuman level of play in Chess, Shogi and Go. In this assignment, our main purpose is using deep Q-learning, a reinforcement learning algorithm, to build a model for Coganh - a Vietnamese Chess. There is no such model for Coganh built before, so we introduce our own way to train as well as evaluate the model. Although there are some aspects need to be improved and require further experiments, we can conclude that the outcome model, based on our measurement, is acceptable at the moment.**

I. Introduction

Our main purpose is using deep Q-learning to build an AI agent to play Coganh - a Vietnamese chess. This report has four main sections: Introduction, Method, Result and Discussion. The Introduction section introduces Coganh, Q-learning and deep Q-learning. The Method section explain the way we apply deep Q-learning while Result section will discuss about our training results and evaluation method. The last section, Discussion, will express some limitations associating with potential improvements.

A. Introduction to Coganh

Coganh, a Vietnamese chess, is a recreational and competitive board game played between two players. The game originated from Quang Nam, Vietnam. Coganh is played on a $5x5$ board with 8 pieces belonging to each player. At the beginning of the game, a player is assigned one color, so as their pieces. Usually, the two common colors are red and blue and the players will be called as red player and blue player respectively.

Similar to Go, a Chinese chess, all the pieces which belong to a player are the same in shape and color. However, during a game, all pieces remain on the board, only their colors are changed, which results in their control is switched from a player to the other one. Therefore, the ultimate goal of a blue (or red) player is to change the color of all pieces into blue (or red). The details of the rules are explained at [1].

We can see that Coganh is fully observable because the agent knows which current state it is. Coganh is also deterministic, which means we can determine next possible actions and know exactly how the current state may change after such an action. As a consequence, we can conclude that this is a game of skill rather than a game of chance so it is possible to build an AI model as our main purpose.

Before moving on to next section, we want to introduce some notations. Firstly, an *episode* is a game played from the beginning to the end (may end after a predefined number of steps). During an episode, steps are made by players. One step is associated with one action of one certain player.

B. Q-learning

Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require a model of the environment (hence ”model-free”), and it can handle problems with stochastic transitions and rewards without requiring adaptations [2]. Game playing is the domain, which is commonly referred to, when illustrating the power of reinforcement learning.

C. Deep Q-learning

Q-learning's goal is optimizing a policy called Q-table using Bellman Optimality Equation [3]. Q-table is a 2-dimensional table which axes are states and actions. By using this table, given a state, we can obtain the optimal action (if the table has been trained optimally) by getting the highest Q-value in the row of the given state.

When the number of states and actions grows significantly, using such table is no longer reasonable. This is when deep learning comes to play by replacing the Q-table by a deep neural networks. The algorithm is then called Deep Q-learning and the neural network is called Deep Q Network (DQN).

Usually, an Deep Q-learning model which make actions is called an agent. Later in this report, we will use AI player or AI agent to refer to the same thing.

II. Method

A. Q-learning hyper parameters

Usually in Q-learning, there are two hyper parameters to consider which are discount factor γ and the maximum number of steps of an episode.

Discount factor γ, which value is a real number in $(0, 1)$, determines the dependence of current state on future states. The larger γ is, the more current state depend on next states (including the next of the next state and so on). We set our discount factor $\gamma = 0.95$.

There are circumstances when the game falls into a loop when players iterate some similar actions. To prevent this loop, we allow each player to play at most 25 steps per episode. After 25 turns, the result is draw if the difference between the numbers of pieces is 0.

B. *Replay Memory*

During iteration through episodes, it is necessary to save the training data to train our model. To have sufficient amount of data to start training in batches as well as preventing the forgetting phenomenon of neural networks, we use the lower bound of 8192 and upper bound of 1048576 samples. If new data appear when the memory reach the upper bound, oldest data will be replaced.

C. *Neural Network Architecture*

Our neural network architecture is shown in Fig. 1, which has 6 layers in total and they are all simple fully connected layers.

There are 25 input units which is the size of flattened board. As for the output layer, to relief the complex of implementation, we decided to encode the actions without considering the rules. We assume that an action is from a cell to a cell, which can be identical as far as the rules are not concerned. While there are 25 cells, the number of actions will be $25 * 25 = 625$.

The optimizer is RMSprop with learning rate $\alpha = 0.00025$. We use two neural networks called target network and training network. While training network is trained frequently at the end of each step of an episode, the target network is, however, updated after a certain number of steps. This practice avoids inefficient training process when updating a model weights based on itself. We update the target network for every 500 steps trained.

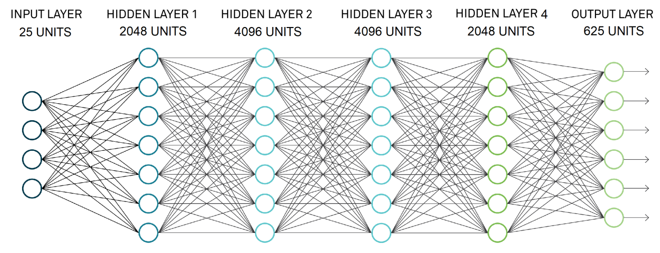

Fig. 1. Neural Network Architecture.

D. *Minimax algorithm*

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal action for a player, assuming that your opponent also plays optimally. It is widely used in two-player turn-based games such as Tic-Tac-Toe, Gomoku, Chess, etc [4]. Actrually, instead of minimax, Monte Carlo Tree Search is used more commonly, which is also used in Alpha Zero [5]. Nevertheless, minimax performs well in this game and therefore, we think that it is worthy to try.

Later in this report, when either minimax player or minimax algorithm is used, it means that we mention about the same thing. Moreover, a minimax algorithm is always associated with a depth. The larger the depth is, the more rational actions a minimax player can make.

E. *Training Algorithm*

As we can see that two players of a Coganh episode are equal, i.e. they have the equal opportunity to win the game. Without loss of generality, we trained the AI player to play well for player red (denoted by -1 in implementation), which is believed to decrease the time for training and the complexity of neural network architecture.

We let the minimax player with $depth = 3$ play against the random player for several episodes with the parameters set up as mentioned. Before each episode, two random instructions will be taken. The first random is used to assign the color to minimax player and random player (red and blue or vice versa respectively). The second random is used to decide whether red player or blue player will make its first action.

For each step in an episode, We carry out these commands in order:

- Record the state, the reward, the action and the result (whether win or tie) of both players to the experience.
- Train the training network and store the history, which is MSE.
- If there are 500 steps played, update target network by the weights of training network.

III. Result

First of all, we introduce new measurements called Winning Rate in Random Games ($WRRG$) and Winning or Drawing Rates in Random Games ($WDRRG$). They are used to evaluate how likely a player will win ($WRRG$) or at least draw ($WDRRG$) in a game versus a random player, in which the random player holds blue pieces. To compute those values, firstly, generate 1000 games between the red player and a blue random player. The $WPRG$ is the percentage of the winning games over 1000 games played. Similarly, $WDPRG$ is the percentage of the winning together with drawing games over 1000 games played.

The reason for introducing new measurements is because of the nature of the game. After training some thousands of games, our AI players perform draw games against minimax with different depths (from 1 to 6) and the result remains the same after a large number of episodes. Therefore, to show the development of the model, it is necessary to introduce new measurements and we come up with $WRRG$ and $WDRRG$.

In terms of minimax players, $WRRG = 95.6$ and $WDRRG = 96.9$ for depth 1 and they are both 100 for other higher depths. As for the AI player, we have trained it for 11000 episodes and both $WRRG$ and $WDRRG$ are approximately 95. In Fig. 2, the AI player's $WRRG$ and $WDRRG$ have grown a lot but still lower than 95 and lower than that of minimax. However, it performs draw games against minimax from depths 1 to 6.

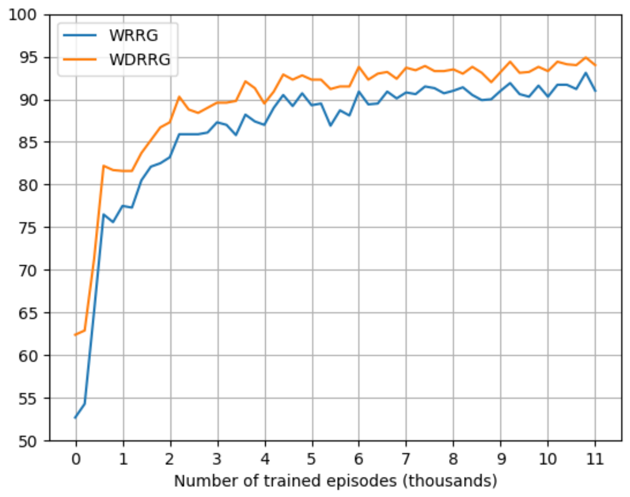

Fig. 2. $WRRG$ and $WDRRG$ over checkpoints of model.

IV. Discussion

Finally, we have completed our model of AI player for Coganh. After all, the model has high $WRRG$ and $WDRRG$ as well as draw results against minimax players. Due to those results, we can claim that our model is as smart as a professional player, who may play draw against minimax players. However, there are still limitations and of course, a lot of space for further research in this topic.

A. Limitations

There are two main difficulties which are resource limitation and the popularity of the game.

In terms of the resource limitation, although a GPU was utilized, the training time is still long and therefore, limits us from trying different training methods as well as neural network architectures.

As for popularity, as a games originated from a local region, Coganh is not popular across the whole country. As a result, it is impossible to gather professional games as neither training data nor measurement.

B. Future research suggestions

Our result at the present is not very remarkable, the AI cannot completely defeat the minimax players and also, it cannot reach 100 regarding $WRRG$ as well as $WDRRG$. This means that there are a lot of space for future improvements. Moreover, if we witness the games played by our model, we can see that the model is not very creative. The improvements may come from:

- **Neural network architecture:** As mentioned, because of limited resource, a very deep, sophisticated neural network is not applied. It is worthy to try a deep residual convolution network.
- **Minimax algorithm:** The minimax algorithm has its own evaluation method. This, and also the depth, could be modified to bring a better minimax player, which will certainly improve training process.
- **Tuning the hyper-parameters:** The parameters can be listed as: discount factor γ, learning rate α, or ϵ in terms of exploration and exploitation trade-off associated with its decay factor, etc.
- **More training episodes:** This is no doubt another promising practice to apply if having sufficient resource.

References

[1] Thy Nguyen Ha Bao, 28 January 2021, *Huong dan cach choi co Ganh, co Chem | Luat choi co dan gian don gian*, accessed 26 July 2021, <https://www.thegioididong.com/game-app/huong-dan-cach-choi-co-ganh-co-chem-luat-choi-co-dan-gian-don-1323649>.

[2] Wikipedia, 12 July 2021, *Q-learning*, accessed 26 July 2021, <https://en.wikipedia.org/wiki/Q-learning>.

[3] DEEPLIZARD, 2 October 2018, *What Do Reinforcement Learning Algorithms Learn - Optimal Policies*, accessed 26 July 2021, <https://deeplizard.com/learn/video/rP4oEpQbDm4>.

[4] Silver, D., Schrittwieser, J., Simonyan, K. et al. Mastering the game of Go without human knowledge. Nature 550, 354–359 (2017). <https://doi.org/10.1038/nature24270>.

[5] GeeksforGeeks, 31 March 2021, *Minimax Algorithm in Game Theory | Set 1 (Introduction)*, accessed 26 July 2021, <https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>.

[6] 28 September 2020, *Day May Choi Tictactoe Bang Deep Learning*, accessed 26 July 2021, <https://codelearn.io/sharing/day-ai-danh-tictactoe-voi-deep-learning>.