COMP30024

ARTIFICIAL INTELLIGENCE

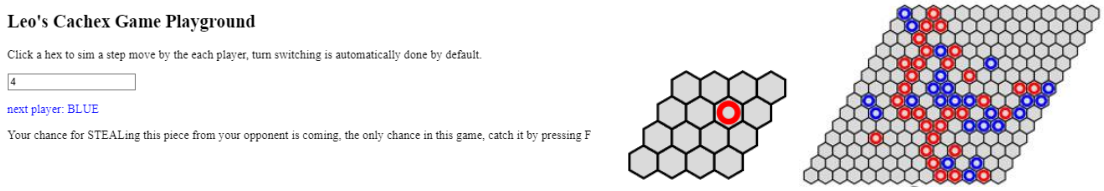
PROJECT PART B

GROUP REPORT

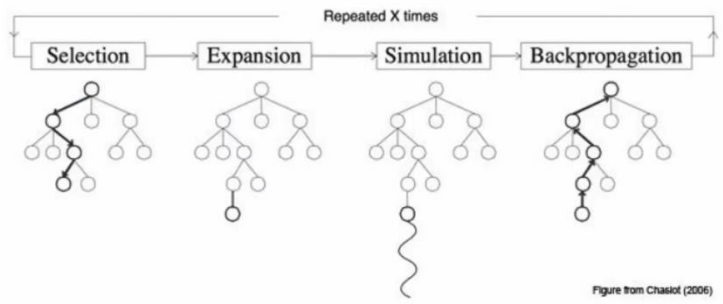
Leo Xinqi Yu, Mengyuan Huang

The original algorithmic approach was chosen to be a self-play adversarial deep neural network -guided Monte-Carlo Tree Search (MCTS), with both policy and value learning techniques stacked on each other. This implementation idea was inspired as I listened to the DeepMind podcast introducing the versions of AlphaGo (Zero), and my mind was settled due to a strong personal interest into the Deep Learning field.

The above approach was fully designed, implemented and tested in all versions except the last one, throughout the whole development timeline. However, in the last minute, the trained neural network was removed from the Monte-Carlo Tree Search because a sudden realisation of the difficulty of deploying the network without using PyTorch, which was the framework that network had been trained on.



In the inception stage, a PvP Cachex playground is developed by ourselves on codepen.io (https://codepen.io/leostina/full/qBxEQNq) , aiming for providing environment familiarise the game rules and tricks, and potentially, providing crucial training data for pattern learning agent. With the playground, 80 games were played between us and all moves were recorded. Meanwhile, articles and papers about a range of Deep Learning techniques were studied or restudied, including CNN, LSTM(RNN), and of course the AlphaGo (Zero) paper by DeepMind team.

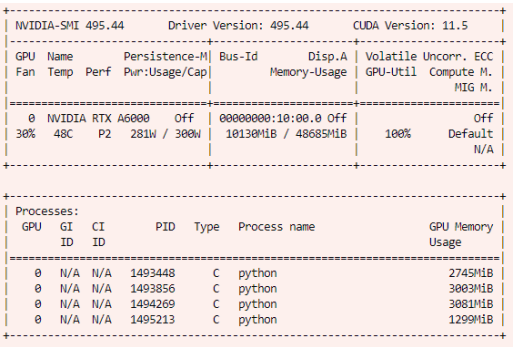


The Monte Carlo Tree Search (MCTS) we utilised, is essentially a heuristic-based Minimax-like tree searching algorithm. It consists of 4 phases, tree traversal (selection), node expansion, rollout and backpropagation. In the selection phase, a node with a best UCB (Upper Confidence Bound, the heuristic value) is selected. For achieving the UCB value of the current node, its child nodes are expanded and simulated (to the terminal state). With having the reward (cost) value from the terminal state, the value is then back propagated upwards in the tree, and updating each node with a new UCB value (Chaslot et al. 2006).

One cool feature about this game board is its symmetric property. One modification to the game implementation is that from both players (red and blue)'s perspective, the same goal is aimed, that is connecting top and bottom sides of the board, with red-color token, and a great implementation difficulty was reduced with this design. Way to achieve this is utilising the symmetrical property of the game (board), flip the board after a player finishes his turn. The blue player simply does not "get know" his true identity except he is marked as the opposite color. In his own board recorder, the turn begins at number 1, and a STEAL action is enabled in his first turn. Symmetrical property also comes in handy when executing self-plays training examples. After a game is played, it's mirrored game along the division line of the two axes, would be immediately recorded, with the opposite player as the winner. Hence double amount of the training examples was generated.

The Neural Network is essentially a function taking the board state as input, and output the value of the current state evaluating the goodness to the current player, and a policy probability distribution on all valid moves given this state. In each training iteration, gradient decent method is applied to minimise the loss (difference) between the predicted outcome and the true result conducted by game simulation. The underlying is for the network to learn predicting what kind of states would eventually lead the current player to win (or to lose). In each iteration of training, a series of self-plays were performed with MCTS as the algorithm to guide and adjusting the weights on predicting policies (Silver et al. 2017). As a new network is generated at the end of the iteration, it is then get put in an arena, completing with the previous network for dozens of games, N-games = 30 as we tuned for a balanced time and fairness trade-off. If the new network's win-rate being greater than 53% - 60% (as board size n decreasing from 15 to 3), it is recorded and the old best-performing network is replaced.

Although quite a few existing frameworks and implementations were learnt from (S Thakoor, M Jhunjhunwala, 2017), or referenced, building and debugging the complete program was time-consuming enough to leave us with only 2 days of actual training. For acceleration purpose, 11 different sized board from 5-15 were trained on 3 rented GPUs, 2 of which are Nvidia RTX3090 (for training board size 5,6,7,11,12,13,14,15) and an Nvidia RTX A6000 for size 8, 9, 10. The trained network weights were harvest from the remote every 5 hours. The resulting AI was clever enough to beat the greedy search agent within only 2h of training, and beating human brainpower after roughly a day.



As the time this report is composed, the training process is still running on the remote GPUs.

REFERENCE

Chaslot G., Bakkes S., Szita I., (2006), *Monte-Carlo Tree Search: A New Framework for Game AI ,* University Maastricht.

Silver, D., Schrittwieser, J., Simonyan, K. *et al.* (2017), *Mastering the game of Go without human knowledge. Nature* 550**,** 354–359.