Applying Machine Learning to Connect Four

Luke Kim, Hormazd Godrej, Chi Trung Nguyen

1. Introduction

We are motivated to pursue a project in game-playing, because it is an insightful way to demonstrate the capabilities of different techniques in AI[1]. We chose to focus this project on the game of Connect Four, since Connect Four is more complicated than those such as Tic-Tac-Toe (which is easily solved by brute force), but it has a smaller state space than games such as Chess and Go. Therefore Connect Four makes a suitable game for a project to experiment with different machine learning algorithms, including those that are more computationally expensive. The purpose of the project is to gain insight into machine learning algorithms that are well-suited for game-playing.

Connect Four is a two-player strategy game on a grid with 7 columns and 6 rows, hence a total of 42 spaces. Each player is associated with a color and takes turns dropping a disc of his or her color in one column. The disc then falls down to the lowest empty cell of the column. The first player to make an alignment of four consecutive discs wins, whether vertically, horizontally or diagonally. If the board is completely filled without forming any alignment, then the game ends in a tie.

Related Work

Connect Four was first solved in 1988 by two researchers working independently: James Allen and Victor Allis[2]. A knowledge-based approach was used by Allis. Both researchers demonstrated that in Connect Four, the first player can force a win through optimal play.

Various different AI approaches have been taken for general game-playing, including deep reinforcement learning, which is how Deepmind's AlphaGo made a breakthrough in the game of Go[3]. DeepMind's AlphaZero subsequently performed even better at the game, using a similar approach but involving only self-play for training[4]. Research has also been done on AI in Checkers with Monte Carlo methods and TD learning, but the results of that project only demonstrated success against random players rather than professional ones[5].

One of the most popular approaches to solve Connect Four is the Minimax algorithm with Alpha-Beta pruning. For example, blog.gamesolver.org[6, p. 3] implemented and recommended such an approach. It appears that further optimization could be made using transposition tables as blog.gamesolver.org[7, p. 7] did, by intelligently caching the outcome of the previous computations.

2. Our approach

Initially, we implemented a Negamax variant of Minimax algorithm to create a solver. However, it did not compute the policy values in reasonable time. We then implemented Minimax with Alpha-Beta pruning to create an AI that can play against a human player. This approach worked surprisingly well, and the program only took a few seconds to make a move.

To apply machine learning to Connect Four, we began by applying some classic algorithms to classify whether a board configuration is a winning or losing position, as a classification problem. We also briefly detoured to use

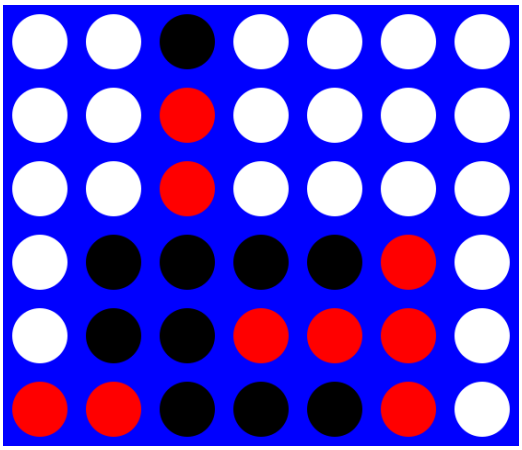
linear regression to predict how many moves it will take to finish a game give a board position. Section 4 describes these models as well as their data and results.

Finally, we implemented the algorithm in the AlphaZero paper and ran an experiment with it. In particular, we designed a neural network that takes a board position as input and gives a probabilistic policy as well as the value, representing who has the advantage, of the position as output. Our experiment was to study the effects of initializing a model with supervised data before training it with the Monte Carlo Tree Search (MCTS) algorithm. Section 5 describes the model and the approach we take for the experiment, while Section 6 talks about our implementation. Section 7 has the results and discussion.

The source code for the project is available at https://github.com/Fenmaz/connect4.

3. Dataset and Features

To solve the classification task of the outcome of a game of Connect Four (win/draw/loss) given a board configuration as input, we used the Connect Four dataset from the University of California, Irvine machine learning repository[8]. It contains unforced positions in the Connect Four board after 8 moves, called 8-ply positions, and the eventual winner. There are 67557 such board positions in total, with 44473 (65.83%) winning, 16635 (24.62%) losing and 6449 (9.55%) drawing positions.



A board position or configuration

Additionally, we used the Connect Four game dataset from Kaggle[9]. This dataset contains 376,641 board positions, where each position is a finished game of Connect Four represented as 42 features for each location on the board. There are 181276 (48.13%) positions where player 1 wins, 180888 (48.02%) where player 2 wins and 14518 (3.85%) drawing positions.

Apart from classifying the game outcome, we also obtain datasets to train our neural network. In order to generate the data we ran several random Connect Four games and recorded the board positions. Then we use the board configuration to send a query to a Connect Four Solver website[10] and scrape the values for the policy, the outcome of the game as well as the number of moves left for the game to finish. Their solver runs on an optimized minimax algorithm (with alpha-beta pruning and transposition table) and therefore had high accuracy. We were able to collect 5994 data points with this method.

Finally, we also ran a pre-trained deep reinforcement learning model to generate 2500 games with more than 30000 board configurations associated with policy and value.

4. Classification algorithms on board configuration

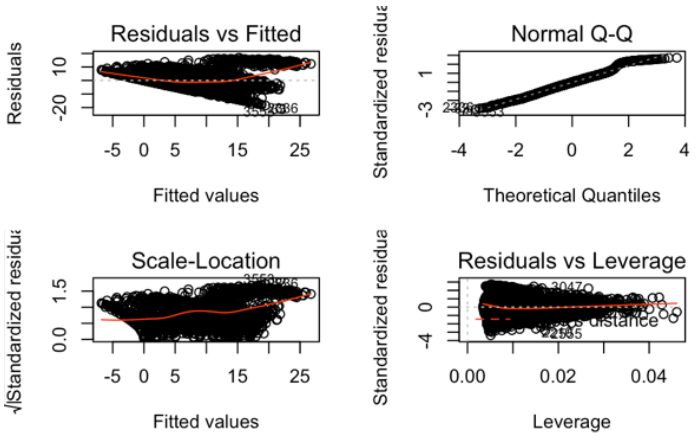
We used logistic regression, support vector machines with linear kernel and gradient boosting machine (GBM) to predict the eventual outcome of the game given a board configuration. Out of the 376,641 data points, we sampled 5000 data to be used as training data and sampled another 5000 to be used for validation.

We used R-packages such as “caret” and “mlbench” to optimize the parameters of the gradient boosting machine, and the one that minimized cross-validation error has 1000 trees, shrinkage 0.001 and a value of 4 for the interaction depth. GBM has the best test set accuracy, classifying 93.34% of the dataset correctly. In

comparison, logistic regression and support vector machines with linear kernel only achieved a test set accuracy of 84.44%.

It appears that the GBM is quite powerful in general as it builds an ensemble of shallow and weak successive trees with each tree improving on the previous. When combined these many trees produce an accurate committee. A better tuning of hyperparameters may have improved the classification accuracy even more.

Linear Regression



A linear regression model was used to predict the number of moves left for the game to finish, assuming that both players play perfectly. The features used were the board positions, the policy values and the outcome of the board. The R-squared value of the fit was low at 0.47. Furthermore, plotting graphs such as Residuals vs Fitted and Residuals vs Leverage reveals that some assumptions of linear regression are not satisfied, such as the constant variance of the errors, with the residuals scattered randomly around zero. In general, the linear regression model does poorly at predicting the number of moves left in a game of Connect Four.

5. MCTS Algorithm and Experiment Design

To play the game with machine learning, we implemented the algorithm from the AlphaGo/AlphaZero papers. The key differences between our algorithm and the papers are:

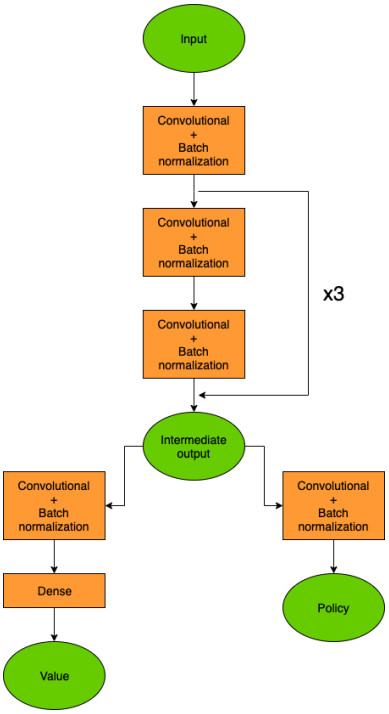
- We use a much simpler convolutional neural network to play the game.
- In the simulation phase, we use the same network to roll out instead of relying on a less accurate approximation, thanks to a smaller model and therefore faster inference.
- Like AlphaZero, we continuously retrain the model with data from self playing.

Inspired by the differences between the AlphaGo and AlphaGo Zero papers, we designed an experiment to study the differences between using labelled examples to initialize the neural network and not. In particular, we run two training programs. The first initializes the network by supervised learning, and the second initializes with random values. We then train both models with the MCTS algorithm for a similar number of iterations, and compare their resulting playing strength. Note that since our model is simple, it overfits easily. To make the evaluation metric meaningful, we use AlphaZero's approach of continuously playing a new model against the previous best and only retaining the best performing model so far. At test time, the models that play against each other are the respective cumulative best models from each training program.

Deep Neural Network

The neural network takes a board position as input and gives policy and value as outputs. The structure is captured in the figure below.

The input is represented as [6, 7, 2] binary arrays where the first two dimensions are the game board and the third for two players. The possible values are 0 and 1, where 1 represented a disc that has been placed. There is one input convolutional layer with batch normalization, and then three residual blocks. Each block has two



convolutional layers interleaved with batch normalization layers, and one residual connection. Each convolutional layer has 64 filters of size 4x4 and stride 1.

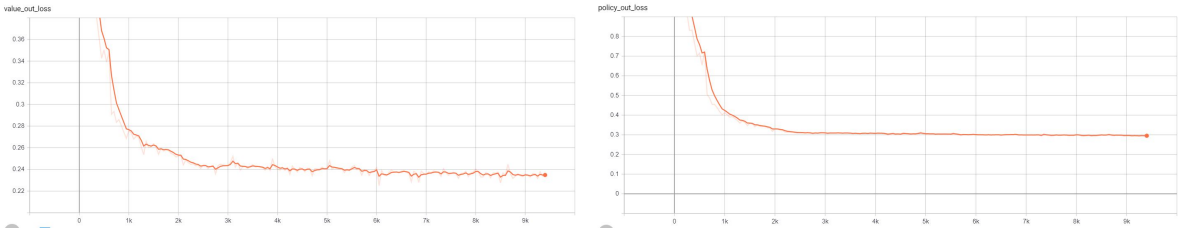
At the end, the network branches into two, one for outputting policy and one for outputting value. The policy branch has one convolutional and batch normalization set of layersm, and the output is an array of length 7, for 7 possible moves at each point in time. The value branch has one convolutional and batch normalization set of layers, and one fully connected layer. The output is a single value in the range [-1, 1], to predict the winner of the board.

6. Implementation

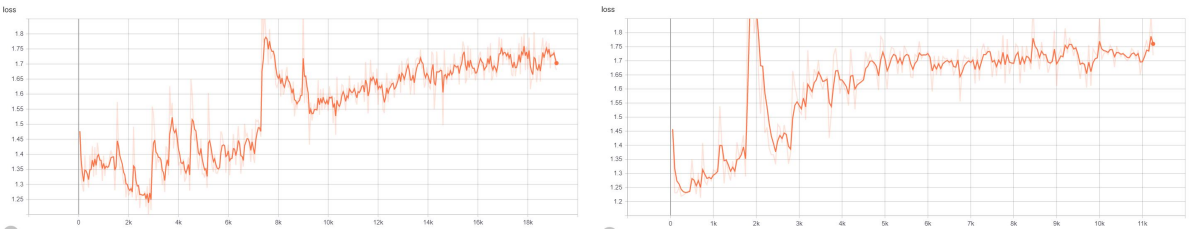
Our MCTS implementation borrows from two open source projects that works with the AlphaZero algorithm and the game Connect Four[11][12].

All training programs were done with the stochastic gradient descent algorithm with a momentum factor of 0.9, mini-batch of size 1024, L2 regularization and a learning schedule of: 10^{-2} from 1 iterations to 500, 10^{-3} from 500 to 2000, 10^{-4} from 2000 to 9000 and 10^{-5} from 9000 iterations onwards. Each move in the selection phase was simulated 100 times, and the constant parameter for PUCT was set to 1. These parameters were selected in accordance with others who have run the algorithm on Connect Four.

In the first program, the model is initialized by training on the labelled dataset of ~30000 board positions with policy and value. The following figures have the policy and value loss curve over time:



Then we started training with the MCTS algorithm, letting the models play against themselves to generate data and refine the model continuously. The overall loss (defined as the sum of policy and value loss) is in the following figures:



7. Evaluation and results

Since new training data is continuously generated, loss over time reflects loss on new data. The loss graphs indicates that the model overfits significantly. This is due to the simple model. For context, the AlphaGo model uses 19 residual blocks, while we use 3.

While comparing the playing strength of models over time is not meaningful due to overfitting, we can compare the cumulative best model. During training we continuously play a new model against the previous one and only keep the winner (>55% over 100 games). The following table has our main results, which is the winning rate of the best model from the unsupervised program against the supervised one. We also provide winning rates of models at different training time for context. All winning rates are evaluated over 100 games, computed as the number of wins over the number of non-tie games. The number of iterations for the supervised program are iterations after initialization is over, at the beginning of MCTS.

Unsupervised program	Supervised program	Winning rate (unsupervised over supervised)
Best	Best	54%
Best	<100 iterations	100%
~4000 iterations	Best	45%
~4000 iterations	<100 iterations	53%
<100 iterations	<100 iterations	0%

Our interpretation of the results is that the model with supervised initialization easily beats the one without after few iterations, but over long periods of training they even out. It takes the randomly initialized model 4000 iterations to reach similar playing strength (45% winning rate). After all training is done, the model without supervision has a slight edge over the one with at 54%, but the margin is small.

Future directions

There are numerous directions the research here can expand on with more resources. Some of the more obvious and probably effective expansions are:

- More complex models, to learn enough to play the game without overfitting
- Compute the winning rate as a more fine-grained function over two dimensions of time
- Separate an evaluation set for the supervised initialization

These are without taking into account better training methods and algorithms.

8. Conclusion

Our experiment with two separate training programs, one with supervised initialization and one without, shows that in a resource-constrained environment, having supervised data helps significantly with speeding up training with the AlphaZero algorithm. The margin in winning rate at the end of our training is small enough that it is hard to tell apart from noise, and we can say that they play at a similar level. Some shortcomings of the experiment includes a simplistic model that leads to overfitting, and the lack of comprehensive evaluation metrics.

References

[1] “Why Teaching AI to Play Games Is Important | Ben Dickson | PCMag.com.” [Online]. Available: https://www.pcmag.com/commentary/362654/why-teaching-ai-to-play-games-is-important. [Accessed: 13-Dec-2019].

[2] V. Allis, “A Knowledge-Based Approach of Connect-Four: The Game Is Solved: White Wins,” *ICG*, vol. 11, no. 4, pp. 165–165, Dec. 1988.

[3] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–503, 2016.

[4] D. Silver *et al.*, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,” *arXiv:1712.01815 [cs]*, Dec. 2017.

[5] I. C. L. Dubel and I. J. Brandsema, “Reinforcement learning project: AI Checkers Player,” p. 9.

[6] “Part 3 – MinMax algorithm,” *Solving Connect 4: how to build a perfect AI*, 19-Feb-2017. [Online]. Available: http://blog.gamesolver.org/solving-connect-four/03-minmax/. [Accessed: 13-Dec-2019].

[7] “Part 7 – Transposition Table,” *Solving Connect 4: how to build a perfect AI*, 24-Apr-2017. [Online]. Available: http://blog.gamesolver.org/solving-connect-four/07-transposition-table/. [Accessed: 13-Dec-2019].

[8] “UCI Machine Learning Repository: Connect-4 Data Set.” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Connect-4. [Accessed: 13-Dec-2019].

[9] “Connect-4 Game Dataset.” [Online]. Available: https://kaggle.com/tbrewer/connect-4. [Accessed: 13-Dec-2019].

[10] “Connect 4 Solver,” *Game Solver*. [Online]. Available: https://connect4.gamesolver.org/en/. [Accessed: 13-Dec-2019].

[11] “connect4-alpha-zero,” Zeta36@. [Online]. Available: https://github.com/Zeta36/connect4-alpha-zero

[12] “reversi-alpha-zero,” mokemokechicken. [Online]. Available: https://github.com/mokemokechicken/reversi-alpha-zero

Contributions

Luke: Luke applied various classical machine learning techniques to our datasets. He also implemented the negamax algorithm, generated data for the Deep RL model to train on and wrote sections of the reports.

Hormazd: Hormazd worked on poster design and poster production. He explored some of the literature on game-playing. He also worked on writing sections of the proposal, milestone and final report.

Trung: Trung collected data, implemented the MCTS algorithm, designed and executed the experiment. He also wrote sections of the reports, manages the project code and write documentation on GitHub.