Connect-4 Machine Learning Report

Joseph Khalaf, Adam Cooper, Kyle Nowak, Will Bray-Cotton, Riley Moorman

Introduction

Connect-4 is a two-player board game where each player takes turns dropping pieces into a 6x7 grid. The first player to connect four of their pieces in a row horizontally, vertically, or diagonally wins. Our goal is to create a machine learning model that is good at playing connect-4. We decided on this problem because we all play connect-4 and find it to be an enjoyable yet relatively simple game.

Problem Definition

We wanted to create a bot that is not just capable of recognizing winning or losing positions but evaluating potential moves and selecting those that maximize its chances of winning. This requires designing and training models capable of learning optimal strategies from a dataset of labeled board states and their mathematically calculated best moves. The input to the system is a representation of a board state, flattened into a vector of 42 features (pos00 to pos41), where each feature corresponds to a cell on the board and can take a value of 0 (empty), 1 (Player 1's piece), or -1 (Player 2's piece). The output is the column index (ranging from 0 to 6) representing the best move for the current player.

Solution

Since Connect 4 is a fully solved game, hardcoding these strategies or using brute-force search algorithms is 100% possible. However, this would require significant computational resources, especially for real-time gameplay. Using machine learning enables the bot to approximate these strategies efficiently by generalizing patterns from data. The bot learns to make decisions that closely mimic optimal play without the need to compute outcomes dynamically during a game. We framed the problem using both supervised and reinforcement learning approaches. Initially, supervised learning was employed, training various models such as MLP and SVM. In order to try to enhance our bots win percent, we utilized reinforcement learning, a popular unsupervised learning technique that rewards or punishes certain moves based on how good they were. This combination of supervised and unsupervised learning approaches allows us to capture a much wider range of different gameplay strategies by the bot.

Data

Our original dataset consisted of rows representing a board state, defined by 42 features (pos00 to pos41) corresponding to the flattened 2D Connect 4 board and a label column that indicated the game outcome: 0 for a tie, 1 for a Player 1 win, and -1 for a Player 2 win. However, this dataset posed challenges for our goal of creating a game-playing bot, as models trained on it learned to predict game outcomes rather than the next moves.

To address this, we created a synthetic dataset. Connect 4 is a fully solved game, meaning the player who goes first can guarantee at least a tie through optimal play. By leveraging this, we designed a new dataset where the board states are still represented the same way, but they are labeled with the mathematically calculated best move rather than game outcomes. To generate this data, we implemented a minimax algorithm with a lookahead depth of 5 moves, creating 100,000 labeled board states. While increasing the depth would significantly improve the data quality and therefore the output of the models, we estimated that it would take multiple days to produce enough observations to train a reasonable model, exceeding the project's timeframe.

Machine Learning Design and Algorithms

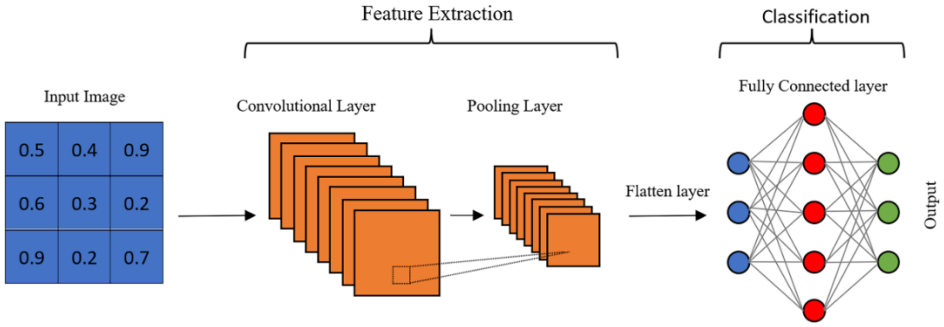
We attempted to solve this problem using 5 different machine learning models. These models being Convolutional Neural Network (CNN), Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), Binary Tree, and Reinforcement Learning (RL).

CNN Model:

Convolutional Neural Networks function by processing data through convolutional layers that apply filters to detect patterns such as edges or specific configurations. These are followed by pooling layers to reduce dimensions while retaining essential features. These extracted features are then passed through fully connected layers to make predictions.

Our model processes the Connect-4 Board as a 6x7 grid to preserve the special relationship between the pieces on the board. This model is made of 3 convolutional layers, each using a 3x3 kernel and no padding in order to maintain the original dimensions of the board. In these layers, we use tanh as the activation function. This is because each board position is labeled -1,0, or 1 representing each player's piece (-1 or 1) or an empty spot (0). If we were to use another activation function such as ReLU, it would ignore one player's pieces as they are represented as a negative, whereas tanh maps real numbers to the range [-1,1] which is perfect for our problem. In addition, we use a 2x2 max pooling layer at each convolutional layer in order to reduce the complexity of the problem while focusing on important pieces. In addition, we have a dropout layer near the end of the model in order to reduce the chance overfitting the model. Finally, the last layer of our CNN model uses a SoftMax function with 7 output nodes. Each of these nodes represent the 7 possible moves to make and which is most likely the correct move in each position.

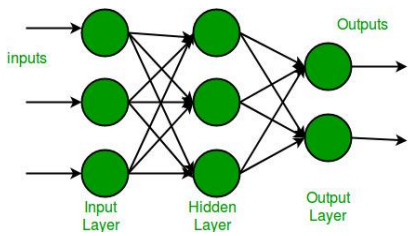
Below is a diagram demonstrating how CNN works. In our model, the input image is a 6x7 array representing the board state, we use 3 convolutional layers, 2 pooling layers, out our output is 7 nodes,



MLP Model:

This model's design is similar in a few ways to the CNN model developed earlier, with some key differences. The primary difference being how it interprets the data—MLPs transform data through multiple layers of nodes that each apply a weighted sum followed by a non-linear activation function, allowing for it to understand complex relationships in the data. One main difference in our MLP model is that it sees the connect-4 board as a 42-element array instead of a 6x7 grid, and is inputted to the model as such. Another key difference is that it uses two large hidden dense layers to capture complex dependencies in the board. Similarly to the CNN Model however, it uses a tanh activation function and outputs with a SoftMax activation function and 7 nodes.

Below is a diagram of an MLP, ours is a much larger version of this with 42 nodes in the input layer, 2 hidden layers of 256 nodes, and an output layer of 7 nodes.



SVM Model:

Support Vector Machines (SVM) are supervised learning models that classify data by identifying the optimal hyperplane that maximizes the margin between data points of different classes. To achieve the highest classification accuracy, we optimized several hyperparameters, including the kernel type (linear, polynomial, and RBF) to map the data into higher-dimensional spaces, and the regularization parameter **C**, which controls the trade-off between margin size and classification error. The input to the SVM model was a flattened 42-feature array representing the current board state of the game. The SVM model then classified the board states into 7 categories, corresponding to the 7 playable columns. To systematically identify the optimal hyperparameters, we employed grid search with cross-validation. The best-performing SVM model utilized an RBF kernel with a **C** value of 10.

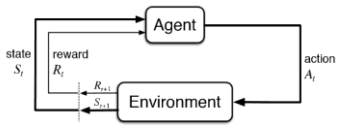
Decision Tree Model:

A Decision Tree algorithm is a supervised learning model that splits data into subsets based on feature values, creating a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a predicted outcome or class. The model works by recursively partitioning the dataset to maximize information gain or minimize impurity at each node. In our model, we tuned several hyperparameters, including the criterion (Gini index or entropy), max depth of the tree, minimum samples per split, and minimum samples per leaf. Similar to the SVM model, the input was a 42-feature array, and we used grid search with cross-validation to identify the optimal hyperparameters. The best-performing Decision Tree model used the entropy as the criterion, with a max depth of 20, minimum samples per leaf of 4, and minimum samples per split of 10.

RL Model:

Reinforcement Learning, unlike every other model we created, is not an unsupervised or supervised learning model, this is because it does not require an initial dataset in order to learn. Alternatively, RL works by creating an agent and an environment. An agent is something that makes decisions in an environment to achieve a goal, and an environment is everything the agent interacts with to make decisions such as state, actions, and rewards. In our case the agent is our connect-4 model and the environment is the game of connect-4 and the reward system. The agent learns by interacting with its environment and receiving rewards based on its actions, where it will continue actions that work towards its goal and ignore those that detract from it. In our model, we reward the agent 10 points for winning a game, 2 points for blocking an opponent's move, -5 points for making a move that allows the opponent to win the game, and 0 points for any other move. It stores all of its decisions in a MLP with an array of size 42 as the input and having 7 node as the output. Given this, we instruct the agent to run 10,000 episodes, or full games of connect-4 while playing against itself in order to improve.

The diagram below demonstrates how the agent and environment of an RL model interact with one another:



How We Made The Models

We created all of our models using python notebooks and using a number of python libraries. The libraries we used include: pandas, numpy, matplotlib, skelearn, tensorflow, and pytorch. We use these to create, train, test, and save our models.

Results

The results of our models in practice are not very good. Although they generally performed well in the classification problem, with an average testing accuracy of 78%, this did not translate well into actual performance. Against random opponents, our models had an average win rate of 72.4%. However, against a hard-coded AI that would make a winning move if it had one, block and opponent's winning move if needed, and choose a random move otherwise, our models had an average win rate of only 12.8%. Our best performer was the Reinforcement model, with a win rate against this sophisticated bot of just 47%. 2 of our models, the SVM and Decision Tree

models, had win rates of 0% against that AI. For comparison, the random AI had a win rate of around 30% against the sophisticated model. As such, our models performed just better than random in this task of Connect-4.

Model type	Testing Accuracy	Win Rate against Random (100 games)	Win Rate against Advance (100 games)
CNN	84%	82%	7%
MLP	69.8%	81%	10%
Decision Tree	79.5%	71%	0%
SVM	81.5%	71%	0%
Reinforcement Learning	N/A	57%	47%

Related work

Previous work on this matter yielded widely ranging results. For example, one research paper (https://github.com/t-brewer/connect4_CNN/blob/master/c4_report.pdf) outlined their method of utilizing a CNN network to train a model to identify a winning board state, under the assumption that this would be useful in training a model to play the game. They created a model that would output a value between 0 and 1, representing its confidence that it would win given a board state. This proved to be a poor assumption, however, as they were only able to produce results of around a 45% win rate against a hard-coded AI.

On the complete opposite end of the spectrum, a different researcher (https://github.com/neoyung/connect-4/blob/master/connect_X.ipynb) was able to utilize Reinforcement Learning to teach a computer how to play Connect 4 very well. They outlined that their approach was very similar to that of the popular AlphaGo bot (https://en.wikipedia.org/wiki/AlphaGo), utilizing a Deep-Q network and experience replay. The results of this experiment were much more promising, with the agent approaching around a 95% win rate against an opponent picking random moves.

While it may be difficult to draw meaningful conclusions from this comparison since the two models played against different kinds of opponents, in our testing of using the CNN model, the rates of win against a random opponent were much lower than 95%. We therefore believe that the Reinforcement Learning model is a much better approach to solving this Connect-4 problem than the CNN model is.

Conclusion

As Connect-4 is a fully solved game, and our synthesized data set leverages the mathematically calculated next best move, this problem is better suited for classical reinforcement learning approaches. Actions taken by an agent are given immediate rewards to either encourage or punish behavior. The use of exploration allows for different strategies to be used, while exploitation encourages the agent to keep taking known good strategies. Neural networks will struggle with this problem as they will need to see more variations of possible game boards to form more complex decision-making patterns.