Machine Learning incl. Deep Learning

Project: Connect Four

1 Connect four

Connect Four is a simple 2-player board game. The players take turns dropping discs into a board of 6 rows and 7 columns. When a disc is dropped into a column, it takes the lowest available space in that column.

A player wins the game when they manage to build a line connecting four or more discs horizontally, vertically or diagonally of their own discs. The game ends in a draw when all slots are filled without a line of the same color.

2 Available implementation and data

A Python class, *Game*, is provided for the board and game status. It also contains methods for changing and evaluating the game state.

The board is stored as a 6 by 7 matrix containing values of 0 (empty), -1 (player 1) and 1 (player 2). It can be access using the *board* attribute. The status has a value of None (game not ended), 0 (draw), 1 (player 2 has won) or -1 (player 1 has won). It can be accessed using the *status* attribute.

This class has an unusual property when an illegal move is made. Instead of forbidding illegal moves, these moves are allowed but result in a loss of the game.

The main methods are

- *play_move(player, column)*: place a piece of "player" into the specified column and update the game status.
- *random_action(legal_only)*: returns a random move while allowing illegal moves or not.
- *winning(player, legal_only, n)*: returns a vector of length seven. Each element is an estimate of the ratio of wins versus losses for every potential move for "player". These estimates are computed using monte-carlo simulation by playing "n" random games.

The methods *random_action* and *winning* have been used to implement two game-play strategies: two random player competing with random moves and two monte-carlo players competing by making the move with the highest win ratio (using *winning*). Other strategies and or combinations of strategies can be implemented as well.

Two datasets have been created containing 10k and 50k games of competing monte-carlo based players. The games are stored in a CSV file in a compressed way. Each record contains a game id, the move made by a specific player and the winner of this game. The meaning of the columns are as follows: "gameid", "moveid", "player", "column", "winner". This format can be converted into a different representation to be used during training. *convert.py* shows a way to convert it into the most straightforward representation.

3 Project Requirement

Even though this game is a simple and solved game for which Neural Networks are a bit overkill, the goal of this project is to build a neural network capable of playing connect four.

3.1 General guidelines

While this project requires some implementation work, having a working implementation only will not be sufficient for a passing grade. The choices made and results achieved will have to be described and documented in a report as well as answers to the questions in Section 6. This report is as important as the implementation.

The report should be at most 4 pages, figures and tables included. Use a 11pt font and include your name and student-number. Make sure that each figure has a caption and is referenced in the text. Clarity of the report is also important: for example axes of plots should labels.

3.2 Implementation

You will build a neural network that can learn to play the game based on the history of previous games. Reinforcement learning is not the goal! This history can be read from the CSV files provided or novel data generated. If novel data is generated, please provide an explanation and the data when handing in the project.

You are allowed to use the provided *Game* class, but do not make changes to this class. While evaluating your project, it is assumed to be exactly as given.

The neural network should be able to predict some kind of outcome that helps with the selection of the next move to play, based on (only) the current situation. It is not necessary to give the entire history as input. The representation of the input and the outcome (and its meaning) is up to you.

Implement a game-play between a random player and a player based on your neural network.

4 Tips

- Convert the CSV data into the representation you want as a pre-processing step and store it to disk in a binary format. For example, use *numpy.save*. This can save quite some time during development.
- The use of external packages is allowed, but please stick to well-known packages and document the requirements in a separate file.
- When using random numbers, explicitly set the seed to your student number when performing the final evaluation. Keep in mind that tensorflow and keras might also use random numbers.
- Think about the parameters of your model, so that the performance and run-time of your training is still reasonable. Maybe start with a smaller subset of the data during the development of the model.

5 Assessment criteria

- Clarity of the implementation. Structure the code in an understandable way by using functions with clear names to explain their behavior, avoid the use of global variables, ...
- Clarity of the report. The goal of the report is the describe implementation choices and the reasons behind them as well as the results obtained. Make a concise and clear exposition.
- Choices made for structure of the neural network. If you tried multiple configurations, explain why those were not selected in the end.
- Choices of the representations evaluated
- Metrics used to evaluate your solution.
- Completeness of your submission: you have adhere to *all* of the requirements.

6 Questions

Answer the following questions for your project. Please read all of them *before* starting with the implementation.

- How did you present your data as input for the network? It is obvious to stick to the representation of the *Game* class and the final outcome, but think and evaluate others.
- How did you represent the output of the network? For example, you could represent the probabilities of the outcomes (win, lose, draw).
- What loss function did you use and how does it relate to the output representation?
- How did you train the network: how did you prepare the data, optimizer, learning rate settings?
- What metric(s) did you use for evaluating the solution?
- Did you do model selection? If so, how?
- How does the performance of a neural network player vs a random player compare to two random players competing? In terms of number of wins, losses and draws.