Report exam Connect Four february 3rd 2025

Canevari Simone

January 2025

1 Analyze and comment the data

1.1 Conformity analysis and statistical distribution of raw data

Initially, I used scripts to check the correctness and consistency of the datasets:

1. One to check whether each database row actually contained only "R", "Y" and "•" and that each row consisted of 42 characters + 1 label character (using $dataset - verifier.py$)
2. One to check *a priori* the statistical distributions of the datasets (using $data - distribution.py$) then compared them with the final results to identify possible inherent biases in the datasets.

The first check confirmed the formatting and consistency of the data so I moved on to the evaluation of the statistical distributions.

Here I show the results related to percentage distributions. For formatting convenience, I have used PD to indicate Percentage Distribution.

PD in test set:

- R: 66.50%
- Y: 22.90%
- •: 10.60%

PD in validation set:

- R: 66.70%
- Y: 23.30%
- •: 10.00%

PD in train set:

- R: 65.81%
- Y: 24.67%
- •: 9.52%

As we can see from the results, the labels are not equally distributed in the datasets, and consequently we can assume a greater precision of the model toward the cases related to the Red player's wins during the classification.

1.2 Raw data visualization

The datasets were in a format that was inconvenient for feature search and modeling purposes. As a first step, primarily for practical reasoning purposes, I used a python script ($raw-to-numerical-matrix.py$) to make each row of a database a table superimposed on the real board game grid like the following:

$$\begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 & -1 & 1 & 0 \end{vmatrix}$$

As can be seen, the characters were changed to numerical factors in case I would later use this format as the basis for a model. The correspondences are as follows:

- 1 = R
- -1 = Y
- 0 = •

2 Design and implement a suitable data pre-processing procedure

Since I chose a neural network as my model, specifically a Multi-Layer Perceptron, I decided to use each individual grid box value as a feature. This resulted in 42 features plus one label.
So starting from the raw data, the only change I made was to convert each value in the datasets into numerical values using the same equivalence table as in section 1, thus obtaining text lines in the databases like the following:

0 0 0 0 0 0 -1 0 0 0 0 0 0 1 0 0 0 0 0 0 -1 0 0 0 0 0 0 -1 1 1 -1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1

With the last value corresponding to the label

Discarded idea(s)

An alternative that I had thought about but then discarded due to computational complexity and difficult adaptation to the actual parameters of the game was to define the following positional variables using grid representation as seen in section 1:

1. horizontal, vertical and diagonal pairs of pawns of the same color.
2. control of the bottom center space [(D,1) in Connect 4 parlance or (4,1) using the numbering of the examination text] known to be the most influential on the probability of victory.
3. the weighted average of the pieces of a player's from the center square [(D,3) or (4,3)] = $\sum_{n=1}^{4}(3 - dist(\mathbf{c}, \mathbf{p_n}))$ with p position of the piece on the game grid $p_n = (x_i, y_j)$ and $c = (4, 3)$ and 3 being the max distance possible from the center.
4. the number of pieces in one or more of the 4 corners.
5. the number of pieces surrounded but only empty spaces

3 Implement, train and evaluate one or more classification models

As a model I chose a Multi Layer Perceptron, in which I matched every possible position on the grid with a neuron. For this choice, I was inspired by the example "*Digit Classification*" from the chapter "*Training Neural Networks*" as in studying it I noticed several similarities with the case of this problem Below I listed the parameters I chose for the model and the reason for the choices.

3.1 NN Settings and reasons of choice

- Input: the network is configured to receive input with 42 neurons (one for each feature in the dataset corrisponding to a position on the game grid as follows). So each input is a vector of 42 numeric values + 1 label.

$$\begin{vmatrix} 6 & 12 & 18 & 24 & 30 & 36 & 42 \\ 5 & 11 & 17 & 23 & 29 & 35 & 41 \\ 4 & 10 & 16 & 22 & 28 & 34 & 40 \\ 3 & 9 & 15 & 21 & 27 & 33 & 39 \\ 2 & 8 & 14 & 20 & 26 & 32 & 38 \\ 1 & 7 & 13 & 19 & 25 & 31 & 37 \end{vmatrix}$$

  The network is configured to input from each position as if they were pixels of a small image on the grid i choose this configuration based on the "*Digit Classification*" example.

- Hidden layers: the network has a hidden layer containing 31 neurons, activated by the ReLU (Rectified Linear Unit) activation function. I chose 31 as number of hidden layer since, searching machine learning forums i found the following formula $n_{hl} = \frac{2}{3}n_{in} + n_{ou}$ [1] and ReLU as activation

[1] https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3

function as the course material indicated it as one of the best-performing activation functions for modern NN applications.

- Number of hidden layers and Batch size: 2 hidden layers and 64 as batch size. For the choice of these values, once I had set the other parameters via literature or course material, I ran an optimization code (*optimizer.py*) that checked combinations of values and then showed the combination that obtained the maximum accuracy.

- Output: the output layer contains 3 neurons, one for each of the 3 possible classes.The activation function used is $softmax$, which transforms the output of the neurons into a probability distribution. Each final maximum probability indicates to which class the input belongs. The activation function I used is $softmax$ since it is the most common function in classifications in the case of NN with non-binary output (thus more than 2 possible classes).

- Loss function: The loss function used is $sparse\_categorical\_crossentropy$, specifically for multi-class classification problems. Suitable when labels are represented as integers. I used $sparse\_categorical\_crossentropy$ after reading up on the internet [2] and seeing it mentioned as a solution for multi-class classification problems when labels (classes) are mutually exclusive and represented as integers.

- Optimizer: the optimiser chosen is Adam, a variant of the stochastic gradient descent algorithm that uses adaptive estimates of gradient moments to update the weights. I read on the internet [3] that Adam is among the most modern and adaptable optimizers, so I read about its general operation and used it.

3.2 Final training and numerical results

After setting all parameters and verifying the databases, I trained the model one final time, making sure to run a high number of epochs but adding an early stop check that stops the training process if the accuracy does not increase for 100 epochs to avoid unnecessarily long runs ($train-and-save-weights.py$). And then using another script that takes the weights obtained as input and classifies a database ($model-application.py$)
Using the optimized setup i obtained:

- Accuracy on the training set: 89%
- Accuracy on the validation set: 84%
- Accuracy on the test set: 82%

The values do not fully comply with the expected standards for modern machine learning models (above 90%-95%), but the model shows little discrepancy between training and validation, indicating good generalization. So I considered the model a good result even if it could be improved.

4 Use suitable data processing and visualization techniques to analyze the behavior of the model; identify which positions in the grid are the most important for winning the game.

As part of the visualization of results:

1. I used a complete code ($train\_classify\_plot\_max\_confusion\_matrix.py$) that performs the training, from which it creates the maximum search plot, and then classifies and then creating the confusion matrix based on the classification results.

2. I then used another independent code that creates, using the output weights from the training, a heatmap superimposed on the connect 4 grid to visualize the most relevant weights positions.

[2] https://medium.com/@shireenchand/choosing-between-cross-entropy-and-sparse-cross-entropy-the-only-guide-you-need-abea92c84662

[3] https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/#h-advantages-of-using-adam-optimizer

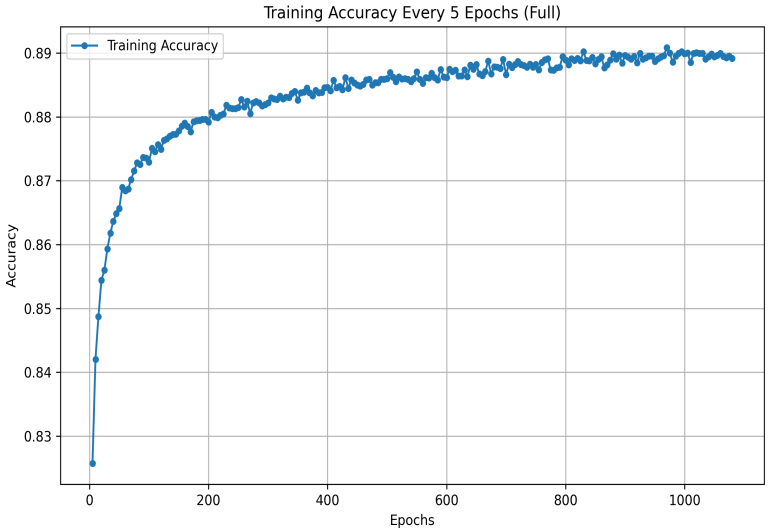

Figure 1: Plot rersult for the search of the absolute maximum.

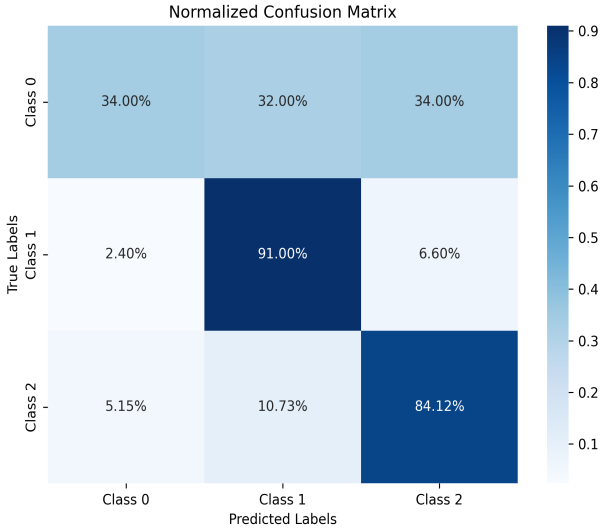

Figure 2: Confusion matrix evaluated on test set with after final training (class 1 = R, class 2 = Y class, 0 = •)

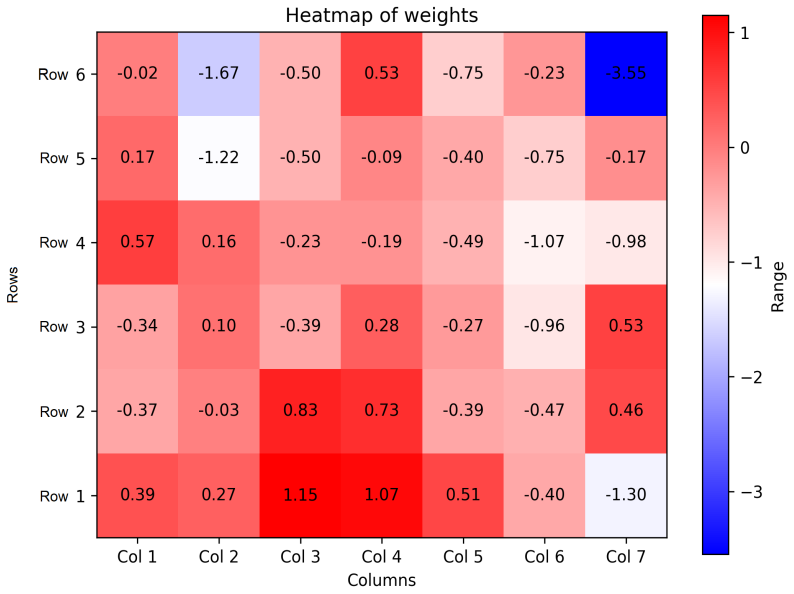

Figure 3: Heatmap of weights on game grid. Positive numbers represent how much the position correlates (with increasing value) more with the probability of winning, negative numbers the opposite.

5 Comments on results and further considerations

With a difference in accuracy between training and validation of 5% and between training and testing of 7% , I consider the results to be sufficiently functional for the purpose, although I am aware that they are not considerably optimal.
Furthermore, from the confusion matrix we can clearly see a clear imbalance between the true positives of the matches won by Red compared to those won by Yellow. However, this is consistent with the distribution of the training set data.
I carried out some tests by artificially modifying the training dataset, removing games and first creating a dataset consistent with the real statistics of the game (i.e. 1% of draw, 49.5% of R and 49.5% of Y) and trying to make train and test set obtaining 98% of accuracy on the train set but 68% on the test set and then trying to make each scenario equiprobable (i.e. 33.3% of R, Y and draw) obtaining 94% of accuracy on the train set but 60% on the test set. I considered these results suboptimal due to the large gaps between the training and test set accuracies since our scope is to balance a high training value with a high classification capabilities.

As a further improvement I could have opted for data augmentation techniques, e.g. by randomly creating new data by generating symmetrical game grids relative to the central vertical axis using samples from the training set (as it is usually done in image recognition problems).

"I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher."