Supervised Learning for Connect Four

Chiara Biguzzi **Enrico Schiaratura** **Francesco Viti**

Probabilistic Methods for Machine Learning
July 29, 2024

Abstract

This study investigates the ability of neural network models to replicate the moves of a pre-trained AI agent in Connect Four. Data processing involves removing duplicates and preprocessing the data for machine learning algorithms. Two model architectures are compared: one using fully connected layers and another using convolutional layers. Models are trained and evaluated, with an analysis of performance metrics and confusion matrices for classification assessment.

Contributions: For the realization of this project we mainly worked together, checking in with each other at every step. We all contributed to the training and testing of the models and the tuning of the hyperparameters. The implementation of the project's initial build was divided into three main parts, with each member assigned a specific section: Chiara focused on the development of the MultiLayer Perceptron structure, Enrico on the Convolutional Neural Network model, while Francesco concentrated on data preprocessing and feature engineering.

1 Introduction

This project aims to explore the performance of deep neural network models in emulating the moves of an AI agent playing Connect Four: it learns from a provided training dataset, comprising of game positions and associated moves. This problem can be seen as a supervised learning task for classification with seven possible classes (0 to 6), each representing a column where players can make moves.

With this aim, we considered two distinct architectures, a MultiLayer Perceptron and a Convolutional Neural Network, since they can both learn complex patterns through training on large datasets. MLP is one of the simplest (yet very powerful) neural networks that takes into account non-linear transformations: after being flattened into a one dimensional tensor, input data is processed through multiple layers of neurons, each fully connected to those in the layers below and above. On the other hand, in CNNs data is processed channel-wise in tensor form through convolutions with specific learnable kernels, which operate on localized regions of the input for feature extraction.

In this report, we will first analyze how we collected and preprocessed the given dataset of game positions and moves. Then, we will describe the implementation and training of both MLP and CNN models. Finally, we will compare the performances of our models.

2 Feature engineering

Upon noticing the scarcity of the training data, we decided to augment it by also considering the symmetric version of each grid and accordingly changing the corresponding move: in this way we were able to double the size of the dataset without introducing new additional data, as a mirrored grid simply represents the same match from the opponent's perspective. At this stage, we realized that this process led to the creation of duplicates in our training dataset, which could introduce bias into our models. We thus decided to remove all the duplicates and finally obtained a smaller dataset consisting of 12878 tuples of grids and moves. During this process we also removed all the duplicates that were already present in the original set of data.

Then, we focused on feature engineering. We first tried to convert each game grid into a 6×7 matrix containing 1's and -1's in the positions occupied by red and yellow tokens respectively and 0's in the empty slots, but realized that this type of input yielded suboptimal results, especially when training MLPs. We thus tried to transform the data into a more effective set of inputs. At first, we created two different 6×7 Boolean matrices, describing the positions of each player's discs in the grid. Next, we tried to include additional features, in the hope of getting better results. In particular, we created two binary matrices of the same dimension marking all the empty slots in the grid that would allow either player to complete an already existing line of three consecutive tokens, as well as a third one highlighting the locations of all the empty cells. We pursued this strategy as we knew that the AI agent was trained to play Connect Four decently. We decided to train some models on both types of inputs, i.e. tensors of size (2, 6, 7) and (5, 6, 7), in order to compare the results of the training procedure, in terms of soundness, loss and accuracy.

3 MultiLayer Perceptron

3.1 Implementation

In our code we proposed two different MLP models, one for each kind of input mentioned above: `fullyconnected_model_ry` and `fullyconnected_model_ryery`. The architectures of these models are very similar, and only differ in the quantity of hidden layers and neurons per layer, which we have both slightly increased for the second model since it takes tensors of bigger size as inputs. Now we give a detailed explanation of our implementation choices. After using a `Flatten` layer to reshape the input into a one dimensional tensor, we defined a sequence of hidden layers, all sharing the following structure:

- `Linear (in_features, out_features)`;
- `BatchNorm1d (num_fratures)`;
- `ReLU`;
- `Dropout(p)`.

At the end, we added a final `Linear` layer with seven outputs, in order to perform the classification. We introduced batch normalization `BatchNorm1d`, which normalizes the output values of the `Linear` layers, and `Dropout` to reduce overfitting and allow for better generalization of the model. We also chose to use the `ReLU` activation function as it performed better than other ones and we opted to employ the `CrossEntropyLoss` as our loss function, since we are dealing with a multiclass classification problem. During all training procedures we used the Adam optimization algorithm instead of the classical stochastic gradient descent to update the network weights, as it converged faster and more reliably; moreover, we added an L2 regularization term to the loss function in order to reduce overfitting and penalize large weight values. We also used the `torch.optim.lr_scheduler.StepLR` function to decay the `learning_rate` by a multiplicative factor `gamma` at intervals determined by the parameter `mu` to increase stability. We tuned all these hyperparameters, as well as the batch size, by trial and error, training our models multiple times over reduced datasets and evaluating their performance on different validation sets with the aim of finding their optimal values.

3.2 Results

We now want to analyze the results obtained after training the two models on the entire dataset of unique samples and finally evaluating them using the test set. It can be noted that both loss functions are steadily decreasing without significant oscillations, suggesting a convergent behaviour of the models. We further observe that, by adding the three extra matrices as input data, we are able to slightly reduce the value of the loss function, especially on the test set, obtaining a smoother decay curve and still avoiding overfitting, as well as considerably increase the accuracy in both the training and test set, reaching a value close to 60%. The reason for this improvement is most likely attributable to the more complex feature engineering. By adding extra features about some potential winning moves for both players, our model is able to overcome its intrinsic unsuitability at recognizing patterns and increase performance, as it can more effectively identify strategic opportunities and threats within the game.

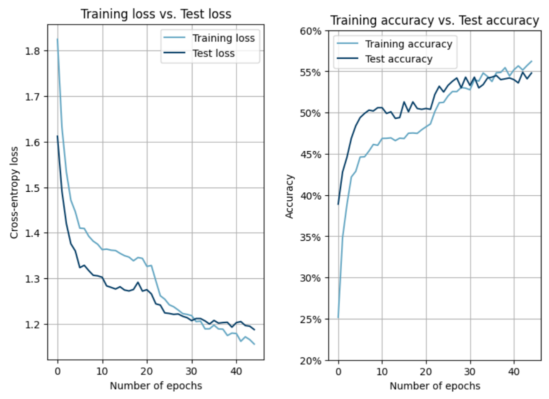

Figure 1: Epoch-wise cross-entropy loss and accuracy for `fullyconnected_model_ry`

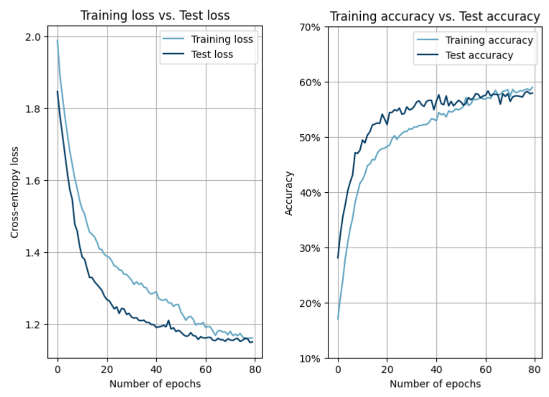

Figure 2: Epoch-wise cross-entropy loss and accuracy for `fullyconnected_model_ryery`

We also decided to examine other performance metrics, displaying for example the confusion matrix for each model, to allow a better visualization of their performance on the test set and a comparison between predicted and actual class labels. We note that in both cases, there is no persistent misclassification between any two classes and that the performance of our models is comparable among all classes. This is particularly evident if we consider the metrics of recall and precision, which are mostly uniform across all seven classes. This indicates that both models are fairly consistent in their predictions and generalize well on unseen data. Finally, we observe that both precision and recall values for all classes are generally higher with the second, more complex model, which, based on our previous considerations, is to be expected.

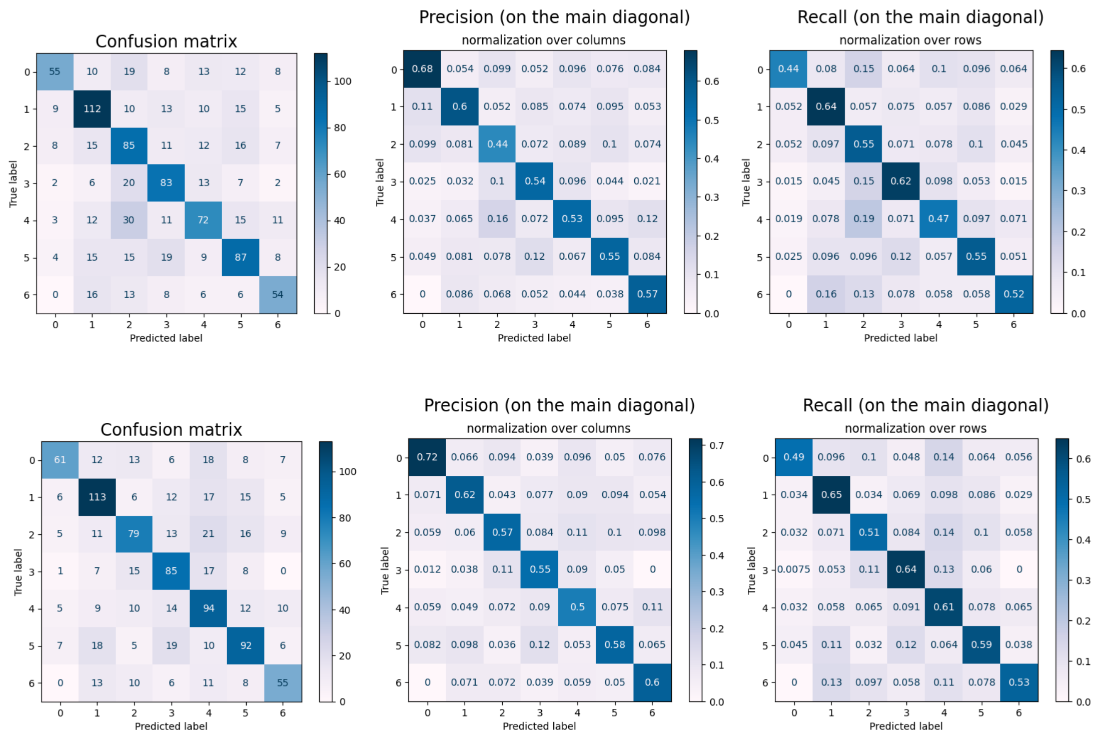

Figure 3: Confusion matrix, precision and recall for `fullyconnected_model_ry`, `fullyconnected_model_ryery`

4 Convolutional Neural Network

4.1 Implementation

This section focuses on Convolutional Neural Networks, which are particularly capable of capturing spatial hierarchies and local patterns through non-linear transformations.

We implemented two distinct CNN models corresponding to the two types of input mentioned in Section 2: `con_model_ry` and `con_model_ryery`. These models share similar structures, with differences mainly in the number of convolutional layers and channels per layer, as the second model handles larger input sizes and thus has a slightly more complex structure. Since we observed that the second type of data yielded better results, we concentrated our efforts on the second model, which we will now thoroughly examine.

Following the input layer, we defined a sequence of four hidden layers, all basically sharing the following structure:

- `Conv2d (in_channels, out_channels, kernel_size, padding)`;
- `BatchNorm2d (num_features)`;
- `ReLU`;
- `Dropout(p)`.

After these convolutional layers, we included two fully connected layers with seven final outputs for multi-class classification. Convolutional layers (`Conv2d`) are the main features that characterize CNNs and distinguish them from fully connected networks. We chose a kernel size of 4 in the first layers to improve the model's ability to understand Connect Four game patterns. The first convolutional layer has a padding size of 2, ensuring that the patterns at the edges are well captured, while a padding size of 1 was used in subsequent layers to reduce the size of the output without losing important information. For the same reasons, we applied a final convolutional layer with kernel size of 3. The relatively high rate of dropout and the batch normalization were included to avoid overfitting, like in the MLP models. All the remaining implementation choices closely follow the methodology described in Section 3: we used the Adam optimizer algorithm, the cross-entropy loss function and employed an empirical approach to fine-tune the hyperparameters values, repeatedly training the model and evaluating it over different validation sets.

4.2 Results

We now examine the outcomes of the training procedure on the complete dataset of the CNN model and the consequent evaluation on the test set. We note that both training and test loss curves exhibit a regular decrease in their values with some oscillations, which however do not affect their overall behaviour. In particular, we observe a significant drop in the value of the training loss after the reduction of the learning rate as well an increase in accuracy, which, however, does not impact the curve of the test loss: this suggests that the model is fitting well the training data but it is also generalizing on unseen ones, avoiding overfitting.

Confusion matrix analysis confirms this thesis, showing a fairly consistent performance, with reasonably high levels of precision and recall, especially on some classes.

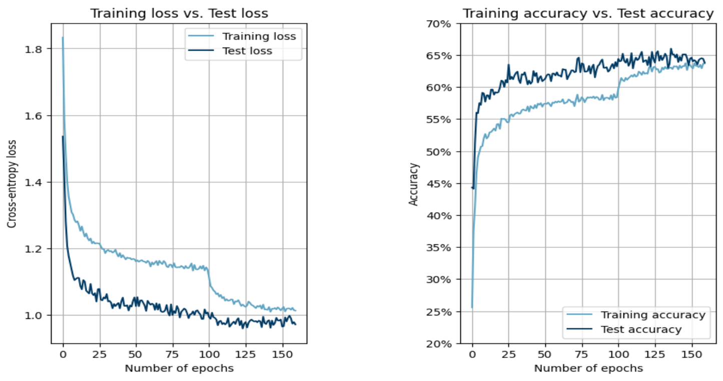

Figure 4: Epoch-wise cross-entropy loss and accuracy of `cnn_model_ryery`

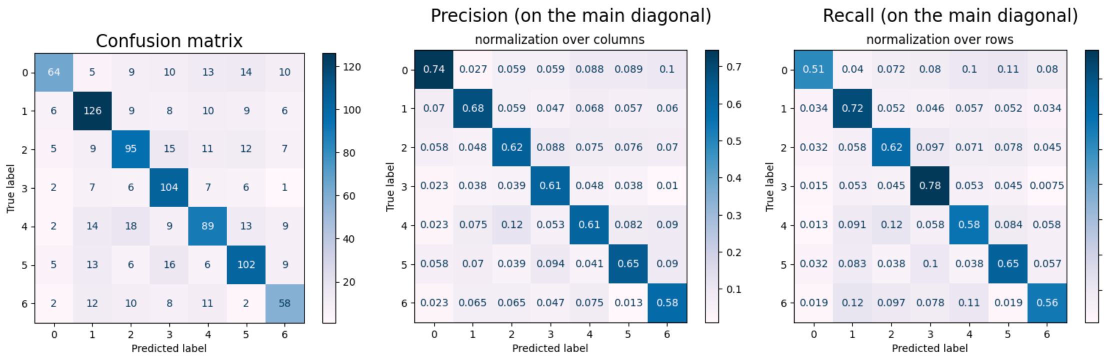

Figure 5: Confusion matrix, precision and recall of `cnn_model_ryery`

5 Conclusions

In conclusion, both models showed effective convergence during training, with the CNN proving to be particularly well-suited for dealing with the complexity of the spatial data inherit in the game, which was expected considering the structure of the network. The effort invested in including additional features led to a significant enhancement in performance for both models, but, on the other hand, resulted in more weights and higher complexity. This improvement was particularly evident in the MLP framework. However, the CNN's capability to capture spatial information and identify intricate patterns ultimately resulted in superior performance compared to the fully connected neural network, both in terms of accuracy and loss.