Probabilistic methods for Machine Learning 2023/24

Task 1

Giacomo Di Paolo, Giovanni Lombardi, Marco Petracci

June 22, 2024

Abstract

In this project, we developed and trained two supervised learning models to replicate the moves of an AI agent playing a variant of Connect Four. We employed two different approaches: a fully connected neural network (FCNN) and a convolutional neural network (CNN). After tuning each model using a small portion of the training set for validation, we compared their performance based on various metrics, including precision, recall, f1-score, and accuracy. We compared the models with both cross-validation and evaluation on the test set.

1 Approach to the problem

First, we analized the supervised dataset and decided to tackle the problem as a multi-class classification problem. The idea underlying this choice is that, at any given position, the AI agent can only take 7 possible actions, corresponding to the 7 columns in which a move can be made. Thus, we have 7 classes, each one corresponding to a column.

We don't know what is the strategy of the AI agent, and so we don't know what features will be relevant for our analysis. This is why we decided to construct deep models, considering the first layers as a way to learn a good feature map.

1.1 Processing of data

1.1.1 Switching of labels

Initially, we needed to convert the board positions into numerical data for our deep neural network. We began by mapping the strings 'X', 'O', and ' ' to the integers 1, 2, and 0, respectively. However, we soon realized it was more effective to differentiate between the current and non-current players rather than simply 'X' and 'O'. Consequently, we started using 1 to represent the current player and 2 for the non-current player.

We believe that this labeling is helpful for our neural network to better understand which player should “make a move”.

1.1.2 Data augmentation

We noticed that our problem has a simmetry along the vertical axis, so we used this property to double the training set. More precisely, after splitting the training set into new training and validation set, for all datapoints in the new training set we added the “flipped” pair. These pairs are created by horizontally flipping the columns of each position and adjusting the corresponding move accordingly. This is because, given the simmetry of the problem, we expect the agent to take specular moves on specular positions.

1.2 First model: Fully connected neural network

We decided to start with a fully connected neural network (FCNN): after running some experiments, we decided to use a network with three hidden layers. From the experiments, this choice looked like a good compromise between the complexity of the network (in particular, the running time of the training procedure) and the accuracy of the model on the validation set. Moreover, adding more layers usually caused the accuracy to drop rather than getting better.

We implemented this model as the class FCModel, and trained it with SGD with Adam optimizer, as seen in class. We only changed the scheduler, using MultiStepLR, to fix the epochs at which the learning rate decreases.

1.3 Second model: Convolutional neural network

Still, the fully connected approach looked like it could be improved. In particular, we wanted to keep track of the fact that our input had a 2D structure, which is not taken into account by the fully connected model that flattens the input before the first linear layer. Thus, we used a convolutional model.

We decided to use a non-squared convolutional kernel in order to give more importance to the information along the horizontal axis. The first reason of this choice is that our input image is a non-squared 6x7 image. The second reason is the fact that, especially in early positions, the spots in the higher part of the columns can’t be occupied. So the game more frequently develops horizontally rather than vertically. Experiments confirmed that using rectangular kernels resulted in better validation scores than square ones. We also avoided using downsampling, since even mutual positions contained a lot of information. Moreover, our “position image” was already of little dimension. Again, experiments confirmed that this was a reasonable idea.

We tried different architectures, but the one proposed by Marco, with 2 convolutional layers and two linear layers, had the best performances. Since we wanted to know the best number of channels for the convolutional layers, we ran a function that cycles through different combinations of the number of output channels of the first layer, the number of output channels of the second layer and the number of output features of the first linear hidden layer. For each combination, with other hyperparameters fixed to reasonable starting values, the function trains the resulting model. Then, we selected those that resulted in the lowest validation loss. This function, called `optimal_channels`, takes approximately 30 minutes to compute using gpu. We ran it, downloaded the results, and then commented out the code.

The units of the second hidden linear layer were fixed to 42 (the image dimension).

We implemented this model as the class ConvModel, and trained it as described above for the FCNN.

1.4 Fixing the best parameters

So far, we have only identified optimal structures for our two neural networks. Next, we created a function to determine the best learning rate, epochs, batch size, and weight decay for each model. This function iterates over different combinations of these hyperparameters, training the models and using early-stopping to find the appropriate number of epochs. Finally, we selected the combination that resulted in the lowest validation loss.

2 Comparison of the two models

Up to this point, we created two models, both of them with a hopefully good enough choice of the hyperparameters. We then needed to evaluate and compare the two models in different ways. In this phase, we avoided using EarlyStopper since we only wanted to use it in the tuning phase, to choose a good number of epochs.

2.1 Cross validation

First, we performed 5-fold cross-validation and computed the average losses across the folds for each model using their respective best hyperparameters. We observed that the best model is by far the convolutional one, even though it is more complex. The results are reported in Table 1.

2.2 Accuracy and other scores on the test set

Finally, we trained both models on the entire training set and compared their performance on the test set using the function `testing`. The convolutional model consistently outperformed

Table 1: Cross validation scores

Model	Num. of param.	Avg. loss	Avg. training time	Avg. accuracy
FCNN	25 291	1,2805	29,28”	49,99%
CNN	95 211	1,1941	32,97”	54,97%

the fully connected model in both accuracy and f1-score. However, it’s important to note that the convolutional model is significantly more complex than the fully connected model: it is deeper, wider, and slightly slower.

We reported some scores in Table 2. We notice that the average scores of the CNN are significantly higher than those of the FCNN.

Figure 1 shows the test loss history over epochs. We can see that the test loss decreases more quickly for the CNN. Furthermore, our models do not seem to suffer much from overfitting, as the test losses stop decreasing after about 15 epochs but never start increasing.

Table 2: Test scores

Model	Test loss	Macro avg. f1-score	Test accuracy
FCNN	1,2490	50,66%	50,70%
CNN	1,1402	56,72%	56,90%

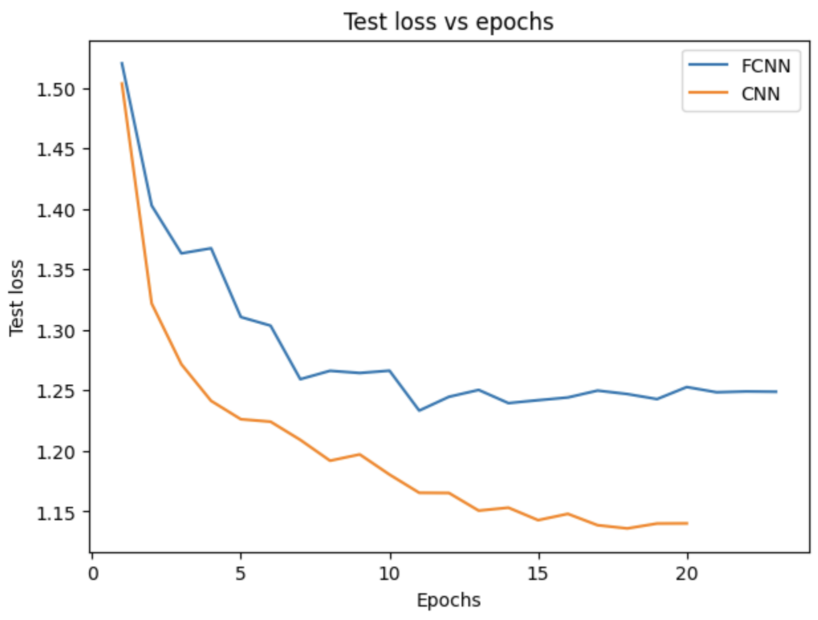

Figure 1: Plot of the test loss of the two models.