Imperial College
London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Deep learning

Author:
Phol Wehaprasirtsak (01859075)
Ines Vanheuverswyn (02469995)
Thomas Aujoux (02464133)

Date: December 19, 2023

1 Fitting our model to the given data

For this project, we will use deep learning to predict high-frequency price changes of two US stocks. In this first part, we will explain the different steps we took to build the model from the data, and how we found the different parameters.

1.1 The Data

To start, `Data_A.csv` is a matrix of dimensions 200000×22, with different types of columns. Column 1 represents the label (direction of midprice alteration (midprice = (bid price + ask price)/2)), encoded with a 0 for decrease and 1 for increase. The Columns 2–22 represent the features, all recorded just prior to the midprice change corresponding to the label. The rows of this file correspond to randomly drawn entries from a larger data set covering the period 1 August 2022 – 22 November 2023, and they can be treated as 100000 independent samples from each stock. No time series structure can be recovered between the rows of the data.

This particular hypothesis on the data will simplify our model for multiple reasons. Indeed, with this hypothesis on the rows we don't need to account for temporal dependencies and simpler models like feedforward neural networks (FNNs) can be used instead of more complex time series models like recurrent neural networks (RNNs). Moreover, this assumption simplifies the training process and statistical analysis as we don't need to account for correlations between data points. Finally, we can use any part of the dataset for training, validation, or for random batching without concern for preserving temporal order.

Secondly, `Data_B_nolabels.csv` contains a 20000×21 array with further 20000 samples (drawn similarly as those in `Data_A.csv`) but with labels omitted.

1.2 Standardization of the Data

The feature standardization is a statistical method that makes the values of each feature in the data have zero mean and unit variance. The general method of calculation is to determine the distribution mean (μ) and standard deviation (σ) for each feature and calculate the new data point by the following formula:

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

There are several reasons why we are using standardization before training a neural network for the classification task. Firstly, it improves convergence. In fact, Neural networks, particularly gradient-based optimization methods like stochastic gradient descent, tend to converge faster when the input features standardised. Moreover, standardization can improve the performance of the model because the model will learn meaningful patterns in the data and make better predictions. Finally, standardized features are more interpretable because their values are in a consistent range. For these different reasons, we chose to use standardization as a pre-processing step.

1.3 Validation Data

In this part and part 1.4, we will explain more concretely the general method adopted to select the parameters for our neural network.

The primary purpose of validation data is to assess how well your model generalizes to unseen data. During training, the model learns to fit the training data, but the real test of its performance is how it behaves on data it hasn't seen before. This prevents the model from continuing to fit the training data too closely and to have overfitting. Validation data is used to tune hyperparameters such as learning rates, batch sizes, and network architecture.

By analyzing the training and validation performance, as we will see in the parts 1.4, 1.5 and 1.6 of the report, you can calculate your model bias and variance. A large gap between training and validation performance may indicate overfitting, while poor training and validation performance might suggest underfitting.

1.4 The Hyperparameters of the Model

We will now discuss the reasons for the choice of the different hyperparameters of the model for the output layer, the hidden layers, and the whole layer architecture.

Output Layer

We will use the sigmoid for the Output Layer. Indeed, the sigmoid activation function is a good choice for the output layer of binary classification tasks, where you want to predict a probability between 0 and 1. The result of the neural network will be in a range of $[0, 1]$, and we can interpret it as a probability that the input belongs to one of the two different classes (the two different classes are the labels in the first column of the dataset as discussed in part 1.1). For prediction of binary classification problem, we decided to adopt the 0.5 as the threshold for the probability. That is, all outputs with value exceeding 0.5 is considered as 1; otherwise 0.

Hidden Layers

For the hidden layers, we will use ReLU (Rectified Linear Unit). ReLU is a widely used activation function for hidden layers because it introduces non-linearity without saturating for most inputs. An activation function is said to be saturating if the output is bounded, which is nowadays seen as an undesirable property for hidden-layer activations as it can cause problems with gradient-based learning. Another strength is that ReLU and its derivatives are mathematically very simple, making them numerically efficient. One potential drawback of ReLU is the ”Dead ReLU” syndrome, where neurons may become inactive during training and never recover. Even if the ReLU used as the hidden layer presents some flaws we will use it because it has already perform well on projects like this.

Architecture of the Neural Network

For the architecture of the Neural Network, we will compare two different types: one with three layers and one with two layers.

The two-layer architecture has different advantages. Firstly, a two-layer architecture is simpler and computationally more efficient compared to a three-layer architecture. It has fewer parameters, so it will result in less training time for the cross-validation. Moreover, a simpler architecture can provide good results without the risk of overfitting as discussed in the part 1.2 of the report. We will compare the result of this architecture with the other one.

The three-Layer Architecture can be also used in this problem. Firstly, a deeper network with more hidden layers can lead to a more complex model and find new relationship between the features of the data set. Moreover, a more complex model introduces non-linearity to the model, which is important in order to find non-linear relationships.

1.5 Stochastic Gradient Descent

In this part, we will discuss our choices of the optimizer and the parameters for stochastic gradient descent.

Adam optmizer

We will now see why we chose Adam as the optimizer for this problem. Firstly, Adam is known for its ability to adapt learning rates for each parameter during training. It computes individual learning rates for each parameter based on the historical gradient information. This particularity used by the Adam optimizer helps to improve the converge of the optimizer. Moreover, Adam is known for the effectiveness large data sets.

The learning rate

The learning rate η indicates how much the model learns in each iteration or epoch during the Stochastic Gradient Descent. A too-small learning rate might prevent the model from converging to the optimal weights, while a too-large learning rate could cause the model to overshoot the optimal solution.
Learning rates typically fall within the range between 10^{-6} and 1. This range provides a guideline for selecting learning rates that balance effectiveness without causing convergence issues.
The diagnosis plot of loss over training epochs during training for a range of learning rates was use to find a good learning rate (Figure 1). Based on the plots we could visualise the properties of the models such as the velocity of the model: has it learned too quickly? (sharp rise and plateau) or is learning too slowly? (little or no change). Are they oscillation in loss indicating a a too large learning rate?(Bengio 2012)

It is apparent that 0.0005 is extremely slow, the loss barely decays over the 30 epoch, and 0.015 is too high considering the oscillations. A good equilibrium is found for the learning rate to be η =0.001.

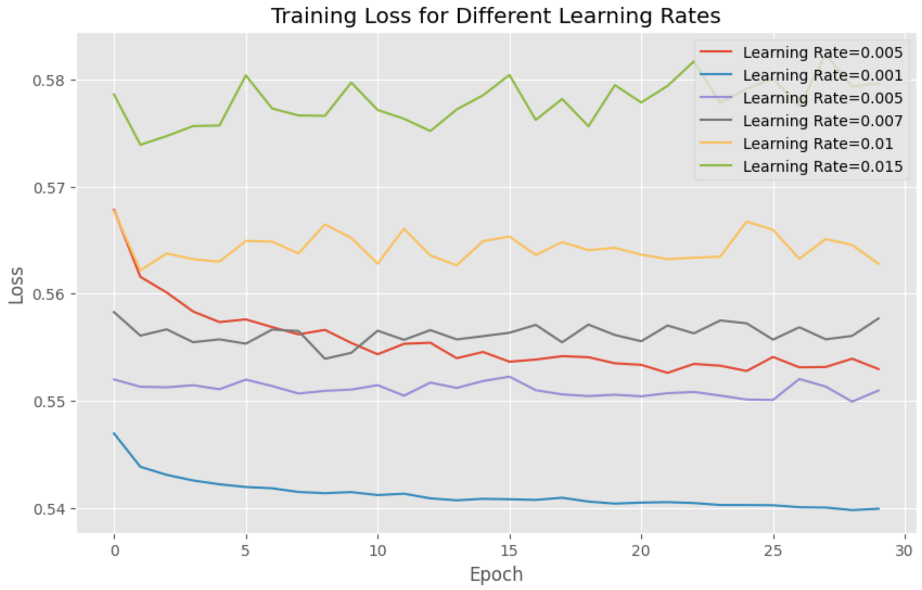

Figure 1: Loss for Epoch Depending on Learning Rates

Size of the Batch

An important choice for the stochastic gradient descent is the size of the batch. The batch size determines how many data samples are used in each iteration of the training process. Firstly, a batch size which is small will produce more noise but can lead to better generalization and reduce the variance of the model. On the other hand, a larger batch size can help us to have stable updates but it may lead to over fitting which is a negative property.

In our project, we have chosen a batch size of 500, indicating a trade-off between efficiency and generalization.

Number of Epochs

We will now explain our choice concerning the number of epochs. The number of epochs determines how many times the entire training data set is processed by the model. If we choose to have a small number of epochs the model will under fit, while using a big number of epochs will over fit our model. Training for an appropriate number of epochs helps balance between under fitting and over fitting. In our project, we have chosen to train for 5 epochs, indicating a trade-off between computational efficiency and generalization.

1.6 Conclusion

All things considered, the neural network performed best for the binary price direction prediction task was a Feed-forward Neural Network(FNN) with all the 21 features as the input

$$\widehat{p} \in \mathcal{N}_3(21, 25, 25, 1; \text{ReLU}, \text{ReLU}, \text{sigmoid}) \tag{1}$$

where the performance metric was ”accuracy” as defined below.

$$\text{Accuracy} = \frac{\text{Correct\_prediction}}{\text{Total\_Labels}}$$

and the neural network (1) yielded the accuracy of the validation data in the range of $\in (73\%, 75\%)$

2 Predict the labels missing using the binary classifier created in Part 1

Data_B.csv was imported and it contains a 20000×21 array with further 20000 samples (drawn similarly as those in *Data_A.csv*) but with labels omitted.

2.1 Method

(1) The features needed to undergo the standardization as explained in Part 1.2.

(2) Using the FNN model (equation 1), price direction labels were predicted in terms of probability.

(3) A function was written to convert all probabilities to be $\in \{0, 1\}$. (threshold of 0.5)

2.2 Result

Our label prediction is obtained as an 1D-array with zeros and ones and was saved as *.txt* file for submission.

References

Bengio, Yoshua (2012). “Practical recommendations for gradient-based training of deep architectures”. In: **arXiv:1206.5533**, pp. 8–9.