Bachelor's Thesis

Bachelor Degree in Informatics Engineering

Bachelor Degree in Mathematics

# High Frequency Trading via Convolutional Neural Networks

*Author*
Miguel Cidrás Senra

*Supervisor:*
Daniel P. Palomar

*Convocatory:*
May 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH Facultat de Matemàtiques i Estadística

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH Facultat d'Informàtica de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH Centre de Formació Interdisciplinària Superior

## Abstract

The objective of this project is to develop and train a CNN capable to trade in a HFT. The methodology followed consisted in design models to forecast particular features of the LOB. The start point was to test traditional methods in finance with the purpose of predicting futures prices. These methods were rejected through experimentation.

Subsequently, a study on ANN architectures was conducted to build models capable of predicting the direction of the mid price. Furthermore, experiments were carried out on different models, testing different representations of the LOB. The results showed how, with proper input and architecture design, a CNN slightly outperforms a MLP in predicting price direction.

Finally, new labels were introduced to detect when the trade has benefits. Thereupon, experiments were conducted to predict these labels. In that case, both models had positive results but CNN did not outperform MLP.

*Keywords: Deep Learning, Convolutional Neural Networks, Multilayer Perceptron, Artifical Neural Networks, Limit Order Book, High Frequency Trading*

*MSC2020:* 91B84

# Contents

# List of Figures

# Acronyms

|  |  |
|---|---|
| **ANN** | Artificial Neural Network. 10 |
| **CNN** | Convolutional Neural Network. 10 |
| **FNN** | Feedforward Neural Network. 10 |
| **GAF** | Gramian Angular Field. 14 |
| **HFT** | High Frequency Trading. 7 |
| **LOB** | Limit Order Book. 7 |
| **MLP** | Multilayer Perceptron. 10 |
| **ReLU** | Rectified linear unit. 17 |
| **RNN** | Recursive Neural Networks. 7 |

# 1 Introduction

High Frequency Trading (HFT) is a type of financial trading that has very short-term investment horizons, of the order of minutes or even less, as opposed to lower frequency trading like daily trading or monthly trading. Given the quick decision-making and the advancement in technology, it is executed in an automated way with algorithms, known as algorithmic trading.

These decisions have to be made based on the past history of limit order book. Practitioners usually use handcraft features based on their experience and then use some simple linear regression for the forecast. Our purpose explore the potential of deep neural networks networks to bypass the need for handcrafting features. Since the data follows a time sequence, the Recursive Neural Networks (RNN) are generally considered the best neural network architectures. However recent studies show that convolutional networks can perform comparably or even better [BKK18], so we are going to explore this possibility.

The project is divided in four main sections. The first section explains the core concepts of the project, the second section describes the transformations applied to the initial data, the third section describes how our neural networks are built, and the fourth section presents the results of the experiments. In the annexes, we extend the description of traditional models and discard a possible custom loss function.

# 2 Theoretical background

This section explains the central theoretical concepts of the project. For this reason, the section defines the Limit Order Books and introduces the models used in the project.

## 2.1 Limit Order Book definition

Limit Order Book (LOB) is a mechanism that facilitates trade, which today has become essential in the operation of many markets due to the fast pace and competitiveness of the financial world. In this section, we provide a LOB description that is common to most markets.

The LOB essential element is an *limit order*, which is an order to buy or sell a given amount of asset at a specified limit price or better. An limit order $x$ can be expressed as $x = (p_x, \omega_x, t_x)$, which was submitted at time $t_x$ with price $p_x$ and size $\omega_x$. The sign of $\omega_x$ indicates if it is a arrangement to sell or buy. In this way, we have two cases:

- If $\omega_x > 0$ is a commitment to sell up to $|\omega_x|$ units of the traded asset at a price no less than $p_x$.
- If $\omega_x < 0$ is a commitment to buy up to $|\omega_x|$ units of the traded asset at a price no greater than $p_x$.

For a given LOB, the units of limit order price and size have some conditions, as there exists two *resolution parameters*:

- The *tick size* $\pi$: the smallest difference between two prices. All orders must arrive with a price $p_x \in \{k\pi | k = 1, 2, \dots\}$.
- The *lot size* $\sigma$: the smallest amount that can be traded. All orders must arrive with a size $\omega_x \in \{\pm k\sigma | k = 1, 2, \dots\}$

The structure of a LOB can be described as a set of two of priority queues, one which consists of active sell limit orders and one which consists of active buy limit orders. The priority of these queues is usually a price-time, so they are ordered according to prices and if the prices are the same, the submission time is compared. The set of all active limit orders in a market at time $t$ can be named as $L(t)$. This set can be partitioned into two sets, each one corresponds to one of the queues:

- The set of active sell limit order.

$$A(t) := \{x|x \in L(t), \omega_x > 0\}$$

- The set of active buy limit order.

$$B(t) := \{x|x \in L(t), \omega_x < 0\}$$

When a limit order $x$ is submitted, a matching process checks if it is possible to match $x$ to some other submitted order of the opposite type (sell against buy and, respectively, buy against sell). If so, the matching happens automatically and the system will attempt to match the incoming order with the opposite order of highest priority. If the previous order is fully satisfied, it will be removed, and if the new order still has remaining volume, the system will try to match again until it is fully fulfilled or no further matches can be made. At this time, if there is still volume in the new order, it will be placed in the order book with the current volume. In general, the trades made immediately at the best price currently available in the market are called *market orders*.

There are some important terms that are used in the context of LOB:

- The *best ask price at time t*: the lowest price among active limit sell order at time $t$. It is the lowest price at it is possible to buy forthwith at time t.

$$a(t) := \min_{x \in \mathcal{A}(t)} p_x$$

- The *best bid price at time t*: the highest price among active limit buy order at time $t$. Respectively, it is the highest price at it is possible to sell forthwith at time t.

$$b(t) := \max_{x \in \mathcal{B}(t)} p_x$$

- The *mid price at time t*: the mean between the best ask price and the best bid price at time $t$

$$p_m(t) := \frac{a(t) + b(t)}{2}$$

In a LOB, it is useful to consider the group of several limit order by their price, to define pairs of price and volume for each price, where the volume is the sum of all size of the limit order with these particular price. After that, for each type of order, the pairs can be arrange from the lowest price to the highest and therefore, defining in this ways price levels and volume levels. These definitions are important as our forecast will rely on them, as our data does not include information about individual orders.

The *ask price levels* definition is:

$$p_a^{(i)} = \begin{cases} a(t), & \text{if } i = 1 \\ \min\{p_x | x \in \mathcal{A}(t), p_x > p_a^{(i-1)}\}, & \text{otherwise} \end{cases}$$

Then, the *ask volume levels* definition is:

$$v_a^{(i)} = \sum_{\omega \in V_a^{(i)}} |\omega| \text{where } V_a^{(i)} = \{\omega_x | x \in \mathcal{A}(t), p_x = p_a^{(i)}\}$$

Analogously, the *bid price levels* definition is:

$$p_b^{(i)} = \begin{cases} b(t), & \text{if } i = 1 \\ \max\{p_x | x \in \mathcal{A}(t), p_x < p_b^{(i-1)}\}, & \text{otherwise} \end{cases}$$

Then, the *bid volume levels* definition is:

$$v_b^{(i)} = \sum_{\omega \in V_b^{(i)}} |\omega| \text{where } V_b^{(i)} = \{\omega_x | x \in \mathcal{B}(t), p_x = p_b^{(i)}\}\}$$

Focusing attention on a time t, we can draw a plot in which we easily observe the ten price levels and their respective volumes.

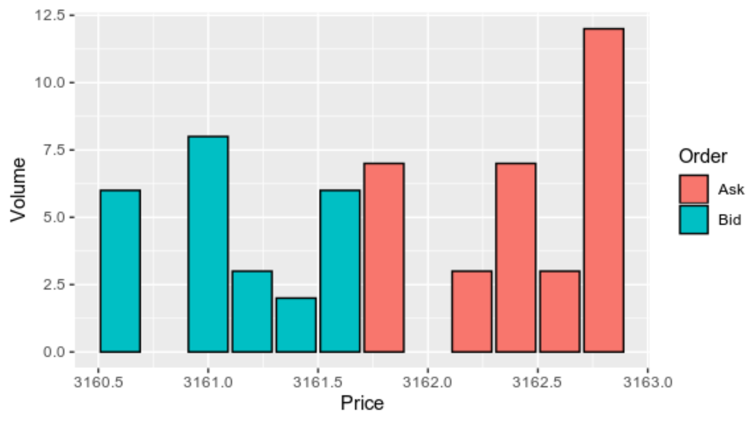

Figure 3: Graphical example of a snapshot

## 2.2 Simple models

We explored two traditional time series analysis methods as baselines for the project, because despite being simple models, they are still used in the finance field.

### 2.2.1 Exponential Smoothing Methods

Exponential smoothing are a class of forecasting methods such that the forecast is a weighted combination of past observations and the weights decay exponentially as the observations get older. This family of methods proves itself as useful, in particular in the finance field.

In finance, it is often helpful to split a time series into several components such as the trend component $T$ (the long-term direction of the series), the seasonal component $S$ (a pattern with a known periodicity) and the error or noisy component $E$ (the unpredictable part of the series). These three components can be integrated in different ways, i.e.:

- Purely additive decomposition: $y = T + S + E$
- Purely multiplicative decomposition: $y = T \cdot S \cdot E$
- Other ways: $y = (T + S) \cdot E$.

Exponential smoothing starts with the trend component, which is divided as a combination of a level term and a growth term. These two components can be combined in different ways and define trend patterns. We explored only no trend, additive trend and multiplicative trend. Our data does not have a clear seasonality. Therefore, no seasonal component was introduced and finally we included the noisy component, either additive or multiplicative.

In total, we get six exponential smoothing methods.

### 2.2.2 ARIMA

While exponential smoothing models are based on a description of the trend and seasonality in the data, ARMA models aim to describe the autocorrelations in the data. They are used to model weakly stationary process in terms of:

- The autoregression (AR) part, which forecasts the variable using a linear combination of past values.
- The moving average (MA) part, which uses past forecast errors.

Since our price time series are no weakly stationary, we replace the data values with the difference between them and its previous values (and this differencing process may have been performed more than once) in order to achieve a stationary time series.

## 2.3 Neural Networks

The Artificial Neural Network (ANN) are a set of learning algorithms of the branch of machine learning, which are inspired by the structure of the human brain. They are useful to model complex non-linear relationships between inputs and outputs.

One view of ANN models is as constructing a function $f$ which approximate an unknown function $g : X \longrightarrow Y$, where $X$ is the input and $Y$ is the output. This function $f$ is defined as a composition of other functions, which can be represented in a network structure connected by directed connections. The basis unit is the neuron, which is a function that receives a input $x$ and has a weight $w$, a bias $b$ and a activation function $\sigma$. The output is $\sigma(w \cdot x + b)$. We have a loss function that we want to minimize.

We explore two classes of ANN: Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN).

### 2.3.1 Multilayer Perceptron

A MLP consist of at least one intermediate or hidden layer, which is placed between the input layer and the output layer. These layers are fully connected with the next one. Since the information propagates layer-by-layer in a single direction from the input layer to the output layer, it is a Feedforward Neural Network (FNN) distinguishing from the RNN.

### 2.3.2 Convolutional Neural Network

In studying an image, instead of flattening the image and feeding it to an MLP, CNNs are built from the idea that nearby pixels are similar and that there are local features like corners and edges. Therefore, we define hidden units that establish a window over the input space and are connected to only a small local subset of the inputs. Now, the hidden units are not connected to all input units.

Following the case of the image, features such as edges may appear in different parts of the input space. So instead of defining independent hidden units that learn different features in different parts of the input space, we can replicate a unit to look at different parts of the input space. To accomplish this, different units have connections to different inputs but share the same weight value. The representation of these shared weights is called a *filter*. The filter shape defines the rolling window shape. A set of filters can be applied to the same input layer, forming a *convolutional layer*.

We can concatenate convolutional layers, and therefore the next layer trains the filters to detect combinations of the detected features. In this way, a hierarchical structure is constructed in which complex features are detected from simple features.

A common structure is composed of the input layer, a combination of convolutional layers and pooling layers, a fully-connected layer and the output layer. A *pooling layer* is a type of layer that reduce the dimensions of the data by combining the outputs of groups of neurons by a single value.

# 3 Data processing

This section explains the details of the data manipulation to get input for ANN models. It refers to the data source, the techniques applied to data preprocessing and the image transformations used in the experiments. The data preprocessing consists in two parts: data cleaning and data normalization.

## 3.1 Data description

Our data comes from the China Financial Futures Exchange (CFFEX). It consists of the LOB of the days from December 2018 to August 2019 when the market has been opened, which is a total of 174. We have access to some instruments of the financial product CSI 300 futures. The units of the prices are index point, each point corresponds with 300 CNY 300. The tick size is 0.2, hence the tick value is 60 CNY, around 7.8 EUR.

The data is structured in snapshots of market data for different instruments of 5 ask price levels, 5 ask volume levels, 5 bid price levels and 5 bid volume levels. The trading hour goes from 09:30 to 11:30 and from 13:00 to 15:15. During this time, for every half-second interval, if there is any activity during this interval, all these activities are combined and we obtain a snapshot of the result. If not, we do not obtain any snapshot.

Different instruments have different expiry dates, which is the last day on which the trading is possible. To define a series of LOB snapshots over the days, we focus on the instrument with the latest expiry date.

## 3.2 Data cleaning

Data cleaning is a important part of the data processing in which data is prepared for analysis by removing or modifying parts of the dataset which are incorrect, incomplete, irrelevant, improperly

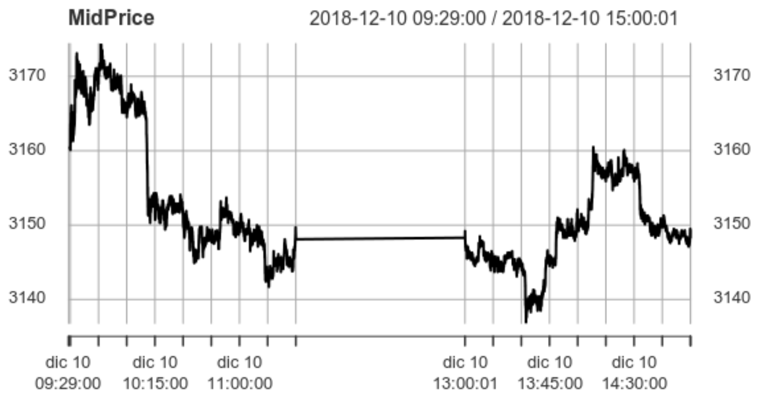

Figure 4: Mid price throughout one day

formatted, or are outliers. Its importance comes from the fact that if the original data is flawed, subsequent analyzes lose precision. However, our data was mostly prepared, so we only need to make three modifications:

- In general, instruments which were actually typographical errors and not actual instruments were removed.
- The time difference between two snapshots was not half-second multiples, but it was always very close to one. Therefore, knowing the origin of the data and knowing the fact that there may be millisecond errors in the time measurement, snapshots time sequence was transformed to always have consecutive difference which are multiple of 500 milliseconds.
- There were no missing values. Nevertheless, as already mentioned, if after half-second there have been no changes in the LOB, a new snapshot is not generated. We decided to replicate the missing snapshots to get a constant time-paced dataset.

## 3.3 Data normalization

In general, all data cannot be directly used for any machine learning task without some kind of preprocessing. At least, we must normalize the input, because:

- For gradient-based algorithms, normalization improves the convergence speed.
- It is required so that all the inputs are at a comparable range so the training is less sensitive to the features scale.

A commonly used optimization technique, such as gradient descent, uses the product of the learning rate times the gradient of the cost function as the step size. As a result, when the features have different scales, the step size can be different for each feature. This can have bad performance consequences because when testing different learning rates, the learning rate may be too small and time consuming to converge, or the learning rate may be too large for one or more features, and it never converges.

We will try to different ways of preprocessing the data.

### 3.3.1 Input standardization

The most common type of normalization scheme is standardization:

$$x_{norm}(t) = \frac{x(t) - \overline{x}}{\sigma_{\overline{x}}}$$

where $x(t)$ is the time series to be normalized, $\overline{x}$ is the mean and $\sigma_{\overline{x}}$ is the standard deviation across all samples.

We normalize each time series of each day separately. In principle, we can not know what will be the value this statistics at the end of the day. In addition, financial time-series usually experiences regime shifts, and because of that using always the same statistics is not appropriate for the complete dataset. Because of that, we are going to approximate these values with the mean and the standard deviation of the previous five days.

### 3.3.2 Stationary

One-day order volume values can be normalized together as they are expected to maintain the same distribution during the day. However, the price time series is not stationary, so it cannot be guaranteed that the statistics of the price values change significantly throughout the day. In particular, this can be seen because prices can reach higher values than any of the above. Further, these assertions are confirmed thanks to the Augmented Dickey-Fuller test.

The solution to convert the price time series to stationary presented in [Tra+20] consist in modifying them to be their percentage difference to the current mid price of the LOB. As a consequence, prices are re-scaled such that the current new mid price is 0. They also use the cumulative sum of the volumes of the price levels as a new time series. In resume, we get the following time series:

- *Price level difference*: the percentage difference of each price level to the current mid price

$$p'^{(i)} = \frac{p^{(i)}(t)}{p_m^{(1)}(t)} - 1$$

- *Mid price change*: the percentage difference of the current mid price to the mid price of the previous time step

$$p'_m(t) = \frac{p'_m(t)}{p'_m(t-1)} - 1$$

- *Accumulative volume*: accumulative volume until each price level

$$v'(t) = \sum_{i=1}^{k} v^{(i)}(t)$$

Analogously, we normalize the time series in the same way as in the previous case: for each day, we standardize each time series with the mean and the standard deviation of the previous five days. Following the notation of [Tra+20], we refer to them by stationary data or stationary time series and to the previous case by raw data or raw time series.

## 3.4 Image Representation

In order to apply CNN to the data, we have to encode sequential snapshots as images to allow the algorithms to "visually" recognize complex features and do the forecast. We explored three approaches.

### 3.4.1 Direct encoding

We directly encode a sequence of consecutive snapshots in one image. With this purpose in mind, we use the $d$ most recent snapshots of the LOB as an input. Specifically, a input if defined as $X_t = \{x_t, x_{t-1}, \dots, x_{t-(d-1)}\}$, where $x_t = [p_a^{(i)}, v_a^{(i)}, p_b^{(i)}, v_b^{(i)}]$ or $x_t = [p_a'^{(i)}, v_a'^{(i)}, p_b'^{(i)}, v_b'^{(i)}, p_m']$, depending in the preprocessing. Time series should have been preprocessed before so that they can be applied to the neural networks.

### 3.4.2 Gramian Angular Field

Following the work [WO15], we implemented the encoding time series Gramian Angular Field (GAF).

Given a time series $S = \{s_1, s_2, \cdots s_d\}$, the first step is to scale it so that all values fall in $[-1, 1]$:

$$\widetilde{s_i} = \frac{2s_i - max(S) - min(S)}{max(S) - min(S)}$$

The second step is to represent the time series $\hat{S}$ in polar coordinates:

$$\begin{cases} \phi = \arccos \widetilde{s_i} \\ r_i = \frac{t_i}{N} \end{cases}$$

where each sample value is mapped into the angular cosine and the time stamp information is represented as the radius. $N$ plays the role normalizer for the span of the polar coordinate system. Then, we can use the angular perspective by considering the sum between each point to identify temporal correlation within different time intervals. The GAF is define as:

$$GAF = \begin{pmatrix} \cos(\phi_1 + \phi_1 & \dots & \cos(\phi_1 + \phi_d \\ \vdots & \ddots & \vdots \\ \cos(\phi_d + \phi_1 & \dots & \cos(\phi_d + \phi_d \end{pmatrix} =$$
$$= \widetilde{S}' \cdot \widetilde{S} - \sqrt{I - \widetilde{S}^2}' \cdot \sqrt{I - \widetilde{S}^2}$$

where $I$ is the unit row vector $\{1, 1, \dots, 1\}$. The GAF can be viewed as a Gramian matrix with a regularization factor.

We use the $d$ most recent snapshots of the LOB and we obtain several time series with $d$ instances. We encode each time series and overlap the resultant matrices. Finally, we obtain a three-dimensional array. The advantages are that it does not need any other normalization and only depends on the current historical data that we have as input -for example, we do not need the estimations for the mean and the standard deviation.The main drawbacks of this method is the data augmentation from each time series of length $d$ to a matrix with $d^2$ elements.

### 3.4.3 Two-channel image

Given certain $r$, $d$ positive integers, we encode a sequence of consecutive snapshots $X_t$ in one image with two channels. The $d$ most recent snapshots of the LOB are the starting point. Analogously to the direct encoding, $X_t = \{x_t, x_{t-1}, \dots, x_{t-(d-1)}\}$, where $x_t = [p_a^{(i)}, v_a^{(i)}, p_b^{(i)}, v_b^{(i)}]$. Now, all prices are scaled to $[1, r]$ and rounded to the closest integer and all volumes are divided by the maximum volume.

We define two matrix $r \times d$ of zeros, where the columns represent the time sequence and the rows the price scale. One matrix will store the ask prices and the other matrix, the bid prices. In order to fill the matrices we proceed as follows: in the column $j$-th, we add the scaled volume at time $t + j - 1$ in the row corresponding to the price at time $t + j - 1$ to the matrix of the same type as the price, i.e., ask or bid.

In the end, we obtain a image of dimensions $r \times d$ with two channels: the Ask channel and the Bid channel. In the same way as the GAF representation, this representation does not need any other normalization and only depends on the current sequence of snapshots.

It is important to notice that we do not encode the stationary time series because

- In case of colliding two prices in the matrix, the sum of accumulative volume has no sense.
- The mid price change does not have a volume, so it cannot appear.
- Since $p_a'^{(1)}(t) + p_b'^{(1)}(t) = 0$ and 0 would correspond to only one row, the result would be equal to scale the price with the other prices of the same time $t$. The main issues is that prices would revolve around the price and there would be no global rise or fall in prices.

# 4 Technical Details

In this section, we explain the techniques and tools used to build and evaluate the models. Therefore, we refer to the general architecture of our ANNs, the description of the labels and the functions that evaluate the models.

## 4.1 Implementation

We use the language R as framework because it is widely used for developing statistical software and data analysis and its capacities are extended through the packages. Among all those that have been used, the following stand out:

- The package *forecast* allows to compute ARIMA and Exponential smoothing models and all these variations conveniently. It also provides us the tools to choose the best ARIMA model.
- We have use the package *keras* for the neural network training. Tensorflow is one of the most complete and compatible libraries for Deep Learning and is linked to an API called Keras.

## 4.2 Initialization

The initial random weights of the models are chosen with the glorot uniform initializer, also called Xavier uniform initializer. For each layer, the weights follow a uniform distribution $U\left[\frac{-1}{n}, \frac{1}{n}\right]$ where $n$ is the size of the previous layer.

## 4.3 Architecture

The FNN architecture can be divided in three parts: input layer, hidden layers and output layer. The input is a image representation of a sequence of snapshots. Because of that, the shape of the input layer depends on this choice. On other hand, the output layer structure depends on the labels codification. For binary classification, the output layer consists of one node and its output value is either 0 or 1. For multiclass classification of $C$ classes, the last layer consists of $C$ nodes with binary output. In our case, $C = 3$. Therefore, the main difference between our MLP and CNN models come from the hidden layers.

MLP may have several consecutive full-connected hidden layers, our MLPs only have one hidden layer.

To build the hidden layers of the CNN, we concatenate convolutional layers and we intercalate some pooling layers. After that, we full-connect the last one with one layer, which is also full-connected with the output layer. In our CNNs, the pooling layers cluster the subset of neurons into the maximum. Because of that, we call them max pooling layers. The application of a convolutional layer reduce the dimension of the input. For this reason, we may add some zeros to the original input with the purpose of conserving the spatial resolution after convolution layer. This technique is called *padding*.

The CNNs of section 5.2.4 are inspired by the philosophy of VGG nets [SZ14a]. Thus, the rules that follow are:

- The convolutional layers have filters $3 \times 3$ and use spatial padding.
- Consecutive convolutional layers have the same number of filters, i.e. we should apply padding.
- After a max pooling of $2 \times 2$, the number of filters is duplicated

In [SZ14b], they add shortcut connections to the ANN. The motivation came from the observation that the error of a model sometimes is increased by adding more layers. However, after stacking more layers to a ANN, the performance should at least be maintained, as the ANN with more layers can learn exactly the same as the original. Therefore, shortcut connections are created to allow the ANN to learn deviations from the identity layer. Following the work in [SZ14b], we also add shortcut connections to skip convolutional layers.

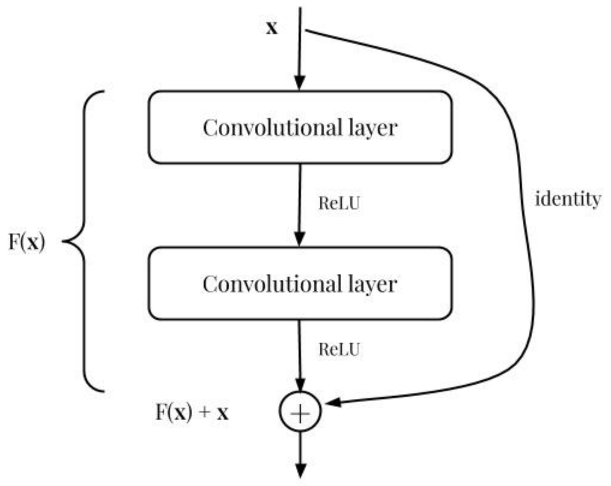

Figure 5: Example of shortcut connection

## 4.4 Activation functions

The activation function determines the relationship between inputs and outputs of a node in a network. The main reason behind using them is to introduce non-linearity in the model. Nowadays, a wide range of functions can be used. Nevertheless, we have only employ the following three:

- *Rectified linear unit (ReLU)* as the activation function for the intermediate layers.

$$\sigma(z) = \max\{0, x\}$$

- *Sigmoid* as the activation function for the output layer in binary classification. It produces similar results to step function in that the output is between 0 and 1, so the output can be viewed as the probability of belonging to a class.

$$\sigma(\boldsymbol{z}) = \frac{1}{1 + e^{-z}}$$

- *Softmax* as the activation function for the output layer in multiclass classification. It guarantees that the output will be in a range of 0 and 1, and the sum of them is 1, thus the output can be interpreted as the probability for each class.

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_k}}$$

## 4.5 Loss functions

Loss function is the function that the neural network is trying to minimize and it measures how good a prediction model does in terms of being able to predict the expected outcome. During the training, the loss is summed over all samples in each batch.

The cross-entropy measures how different two distributions are. Its formulation depends on the number of classes in the classification problem and can be deduced from the minimization of the log likelihood [Bis07]. We denote the ground truth vector by $y$, while $\hat{y}$ is the predicted label distribution.

- *Binary cross-entropy*: when we consider the problem involving two classes in which we consider a network with a single output.

$$E(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

- *Categorical cross-entropy*: when we consider the problem involving more than two mutually exclusive classes in which we consider a network with an output for each case and the target data has a one-hot encoding scheme.

$$E(y, \hat{y}) = -\sum_{j=1}^{C} y_i \log \hat{y}_i$$

We also explore the *focal loss*, introduced in [Lin+17].

$$E(y, \hat{y}; \alpha, \gamma) = -(y\alpha(1 - \hat{y})^{\gamma} \log \hat{y} + (1 - y)(1 - \alpha)\hat{y}^{\gamma} \log(1 - \hat{y}))$$

where $\alpha$ is a weighted factor and $\gamma$ is a parameter such that $\alpha \in (0, 1)$ and $\gamma \geq 0$. The idea behind this function is to add a factor to the cross-entropy criterion in order to reduce the loss of correct predictions, putting more emphasis on incorrect predictions. This factor is regulated by *gamma*. In fact, when $\gamma = 0$, we obtain a weighted binary cross-entropy:

$$E(y, \hat{y}; \alpha) = -(\alpha y \log \hat{y} + (1 - \alpha)(1 - y) \log(1 - \hat{y}))$$

## 4.6 Learning algorithm

The weight of the models are updated by an algorithm based on mini-batch gradient descent. This means that splits the training dataset into small batches that are used to compute the updated weights.

In particular, we use *ADAM* because it is a very popular algorithm and it has a good performance in neural networks with similar architecture [Bas+18]. The algorithm core ideas are to incorporate the previous updates in the current change and to adapt the learning rate depending of the evolution of the error.

## 4.7 Labels

A set of discrete options must be constructed from our data to be used as a target for our classification models. Therefore, we define labels. The most common ones are based on the mid price change, as in [Nta+17]. Although the mid price is not a real value and cannot guarantee the immediate execution of any order if placed at this exact price, forecasting its upwards or downwards movement provides a good estimate of the price of future orders.

Simply using the inequations $p_m(t+k) > p_m(t)$ or $p_m(t+k) < p_m(t)$ to determine the direction of the mid price would introduce a large amount of noise, since even the smallest change would be recorded as a movement. To avoid this issue, an average filter was applied in a future window $h$:

$$m_a(t) = \frac{1}{h} \sum_{i=1}^{h} p_m(t+i)$$

The thresholds $\alpha_1, \alpha_{-1}$ are defined in order to determine how significant is the change of $m_a(t)$ value and if it must be label the movement as upward or downward. Cases which thresholds are not satisfy are considered as having no price movement. The labels for describing the movement are denoted by $l(t) \in \{-1, 0, 1\}$, where $t$ denotes the time step and $\{-1, 0, 1\}$ denote downwards movement, no movement and upwards movement. The resulting label is:

$$l_m(t) = \begin{cases} 1, & \text{if } \frac{m_a(t)}{p_m(t)} > 1 + \alpha_1 \\ -1, & \text{if } \frac{m_a(t)}{p_m(t)} < 1 + \alpha_{-1} \\ 0, & \text{otherwise} \end{cases}$$

Some works, as in [ZZR19], also apply the average filter to the past mid prices:

$$m_b(t) = \frac{1}{h+1} \sum_{i=0}^{h} p_m(t-i)$$

However, as in [Tra+20], we found that replacing $p_m$ value with $m_b$ is an oversimplification of the labels, making the problem significantly easier due to the slower adaptation to sudden price changes.

We also introduced two new labels, this time focused on detecting only upward price movements. The available information does not allow us to infer the real price, however, if the future best bid price is higher than the current best ask price, the price has necessarily had to increase. Detecting this price change also allows a simple strategy to evaluate the forecast in which a buy order is made at time $t$ and then a sell order is made between time $t+1$ and $t+h$. Following the previous notation, the new labels for describing the movement are denoted by $\{0, 1\}$, which denote no movement and upwards movement.

The condition to take into account is given a time $t$, to detect if there is some $k \in \{1, \ldots, h\}$ such that $p_a^{(1)}(t) < p_b^{(1)}(t+k)$.

The first label is based on counting, for a certain horizon $h$, how many $k$ accomplish the condition. We define a function that expresses the frequency with which the condition is fulfilled in a horizon $h$:

$$u_1(t) = \frac{1}{h} \#\{k | k \in \{1 \ldots h\} \text{ and } p_b^{(1)}(t+k) > p_a^{(1)}(t)\}$$

Using this function, we can define the label:

$$l_1(t) = \begin{cases} 1, & u_1(t) > \tau_1 \\ 0, & \text{otherwise} \end{cases}$$

Some threshold $\tau_1$ is defined in order to differentiate more the two classes.

The second label takes to account the maximum difference between the current best ask price and the future best bid prices:

$$u_2(t) = \max_{1 \leq k \leq h} \{\max\{p_b^{(1)}(t+k) - p_a^{(1)}(t), 0\}\} = |\max_{1 \leq k \leq h} \{p_b^{(1)}(t+k)\} - p_a^{(1)}(t)|_+$$

Analogously, some threshold $\tau_2$ is defined in order to differentiate more the two classes. The resulting label is:

$$l_2(t) = \begin{cases} 1, & u_2(t) > \tau_2 \\ 0, & \text{otherwise} \end{cases}$$

The three labels are not only different, but also the thresholds are:

- $l_m$ is very sensitive to small changes of $\alpha_1$ or $\alpha_{-1}$.
- In the $l_1$ case, two thresholds generate different labels only if their difference is at least $\frac{1}{h}$.
- In the $l_2$ case, the difference must be at least the tick size.

### 4.7.1 Codification

The binary labels are encoded with a binary variable. This is not possible with multiclass labels, so we use One Hot Encoding and it is defined a new binary variable is added for each unique label value. In the case of first label, we need three binary varibles to encode it.

## 4.8 Performance metrics

We define the functions used to evaluate the superiority of the different models.

The function that evaluates the predictions of traditional models is:

- *Root Mean Square Error*: which measures the quadratic means of the difference between two variables.

$$RMSE = \sqrt{\frac{1}{N} \sum_{j=1}^{N} (y_j - \hat{y}_j)^2}$$

  where $N$ is the number of observations.

To evaluate the quality of the mid price label prediction, we have used at least one of the following functions:

- *Accuracy*: which measures the average number of correct predictions.

$$Accuracy = \frac{1}{n}\sum_{k=1}^{C}\sum_{x|g(x)=k} I(g(x) = \hat{g}(x))$$

  where $I$ is the indicator function and $C$ is the number of classes.

- *Min F1-score*: is minimum of all F1-score of all classes. F1-score is defined as the harmonic mean of the precision and the recall.

$$F1_i = 2\frac{P_i \cdot R_i}{P_i + R_i}$$

  where

$$P_i = \frac{TP_i}{TP_i + FP_i}$$
$$R_i = \frac{TP_i}{TP_i + FN_i}$$

  where $TP_i$, $FP_i$ and $FN_i$ respectively represent true positives, false positives, and false negatives of the $i$-th class.

In addition, we defined another approach to evaluating labels from a financial point of view which is to evaluate the performance of labels with a trading strategy.

Given a LOB of a day of $N$ snapshots, $\hat{y}$ the predictions for the day, a horizon $h$ and extra price $\rho$ ($\rho$ must be a positive multiple of the tick size), we can define a simple algorithm that compute the profits of the approximation of a simple trading strategy. It is an approximation because we assume that the first order is executed instantly, we do not take into account the cost of trading and we are not always sure that the second order we place has priority.

The description of the algorithm is: if the models predict that the price goes up, we execute immediately an ask limit order of size 1. We send an bid limit order of size 1 and a price increased by $\rho$ index points. After a certain period of time, if the order is not executed, it will be canceled and a size 1 market order will be sent. Its translation to pseudocode is:

```
$cumsum \leftarrow 0$
for t := 1 to N-h-1 do
  if $\hat{y}(t) = 1$ then
    $new.price \leftarrow p_b^{(1)}(t) + \rho$
    $fut.price \leftarrow \max_{1\leq k\leq h}\{p_a^{(1)}(t+k)\}$
    if $fut.price \geq new.price$ then
      $cumsum \leftarrow cumsum + \rho$
    else
      $cumsum \leftarrow cumsum - new.price + p_a^{(1)}(t+h+1)$
    end if
  end if
end for
return $cumsum$
```

In consequence, given a set of predictions for different days, we define as *trading score* to the mean of the output of the algorithm these prediction.

# 5 Experimental results

In this section, we present the results of the experiments we have done. We show every technique that we have applied, the motivations of them and the results. The results are summarized in figures, which show the performance of the models. The benchmark is represented by a red dotted line.

In general, we present the results of two metrics: Accuracy and Min F1-score. Although we are usually guided by the accuracy, Min F1-score serves to verify that the accuracy is not misguiding. An extreme example is the case of having three balanced classes and an accuracy of $\frac{2}{3}$. At first, it might seem like a better model than a random prediction, which has a accuracy of $\frac{1}{3}$. However, this model may be a random prediction between two of the three classes. Since the minimum F1 score would be 0, it would clarify that it is a bad model.

When the number of parameters of an ANN is shown, it has been calculated assuming that the input came from the raw time series. Since the difference is not significant, we decided not to include both numbers for the sake of clarity.

## 5.1 Performance of non-neural network models

Our first approach was to try to short-term forecast the mid-price, using the traditional models and having as input the past mid price values.

We split the data as follow:

- The training set is the mid price of the first ten consecutive days.
- The test set is mid price of the eleventh day.

For the first evaluation, we made a prediction for the horizon 10 time steps into the future and rounded it to the closest multiple of half a tick size. The prediction is a possible mid price thanks to the rounding. Since some of the most simple forecasting methods can be comparatively effective to more complex ones, we chose as benchmark the naive forecast that always predicts the last observation. We measured the $RMSE$. The results of figure 6 show that the methods failed to predict the mid price value.

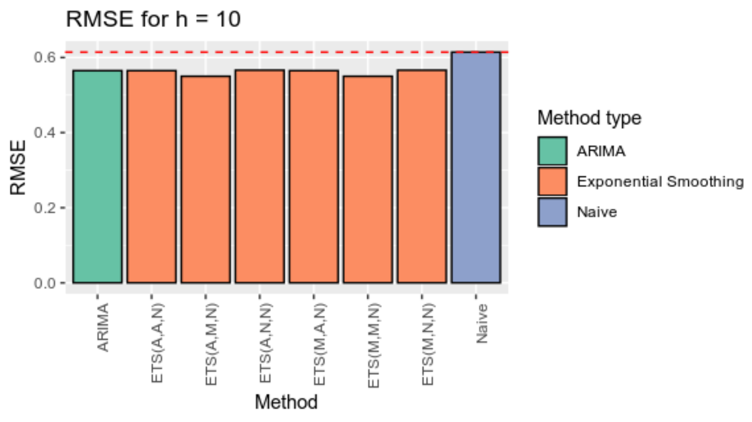

Figure 6: RMSE comparison of simple models

For the second evaluation, we transformed the prediction into the $l_m$ labels. We set $\alpha_1, \alpha_{-1}$ for each day such that $l_m$ was balanced. We measured the minimum F1-Score of each three labels and the benchmark is the random prediction. The results of figure 7 show that neither of these methods outperform the random prediction.

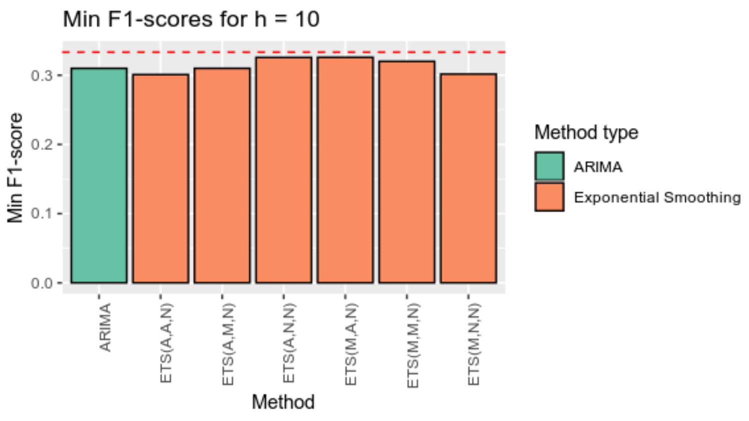

Figure 7: Min F1-score comparison of simple models

In conclusion, traditional time series analysis methods had failed to capture the complexity of the LOB market adequately. This experiment convinced us to address the classification problem directly and not try to predict the price, as we hoped to reduce the noise of the problem.

## 5.2 Performance of MLP and different CNNs with a Mid price based label

We conducted experiments to observe the neural network performance in the prediction of the label $l_m$ based on the direction of the mid price in an horizon h. We decided to set the horizon to $h = 10$ and we set the thresholds $\alpha_1, \alpha_{-1}$ such that the classes are balanced for each day. However, if we had used all of the label values over the course of a day to define thresholds, we would be encoding future information on the label. Thus, the thresholds are defined to balance the labels of the previous five days. This approach for $\alpha_1, \alpha_{-1}$ is common for the rest of the experiments. The loss function is the categorical cross-entropy.

We used early stopping to avoid the overfitting. Thus, the data was always split in training set, validation test and test set and after one epoch, we evaluated the error on the validation set. If the error had not been reduced after two times, the training was stopped.

### 5.2.1 Experiment 1

The first experiment was conducted to find out which normalization had the best performance. Raw time series and stationary time series are represented by images using direct encoding. For this experiment, we set $d = 50$, so the image represented 50 consecutive days. This experiment also explored three forms of filters and because of that, we trained a neural network for each type of filter. The benchmark was the random guess.

We set the maximum number of epochs to 20 and we split the data as follow:

- The training set consisted of the first 40 consecutive days.

- The validation set consisted of the following 10 consecutive days.
- The test set consisted of the following 10 consecutive days.

We trained four ANN:

- MLP with a hidden layer of 128 neurons. It had 128257 trainable parameters.
- The first CNN was adapted from [Tra+20] and its filters were of the form $1 \times f$. It had 181553 trainable parameters. The structure was:
  1. Convolutional layer with 16 filters of size $1 \times 10$
  2. Convolutional layer with 16 filters of size $1 \times 10$
  3. Convolutional layer with 16 filters of size $1 \times 8$
  4. Convolutional layer with 16 filters of size $1 \times 6$
  5. Convolutional layer with 16 filters of size $1 \times 4$
  6. Fully-connected layer with 32 neurons
  7. Fully-connected layer with 3 neurons
- The second CNN was adapted from [Zhe+15] and its filters were of the form $f \times 1$. It had 148977 trainable parameters. The structure was:
  1. Convolutional layer with 16 filters of size $5 \times 1$
  2. Max pooling of $2 \times 1$
  3. Convolutional layer with 16 filters of size $5 \times 1$
  4. Max pooling of $2 \times 1$
  5. Fully-connected layer with 64 neurons
  6. Fully-connected layer with 3 neurons
- The third CNN used the most common form of filter: square filters. It had 102,049 trainable parameters.
  1. Convolutional layer with 16 filters of size $3 \times 3$
  2. Convolutional layer with 16 filters of size $3 \times 3$
  3. Max pooling of $2 \times 2$
  4. Convolutional layer with 16 filters of size $3 \times 3$
  5. Convolutional layer with 16 filters of size $3 \times 3$
  6. Max pooling of $2 \times 2$
  7. Fully-connected layer with 128 neurons
  8. Fully-connected layer with 3 neurons

The results of figures 8 and 9 confirm that the transformation into stationary time series of the data significantly improve performance. Consequently, direct time series coding is constructed from stationary series.

On other hand, no CNN had a significantly better performance than the random guess. In particular, the three CNNs and the random guess had a very similar Min F1-score. However, the third CNN seemed to have the best performance, as its accuracy outperforms the other CNNs. Since this CNN also was the one with the most traditional filters, the results convinced us to use square filters.

In addition, since MLP outperformed the rest of the models and the benchmark, it became the new benchmark for the CNNs.

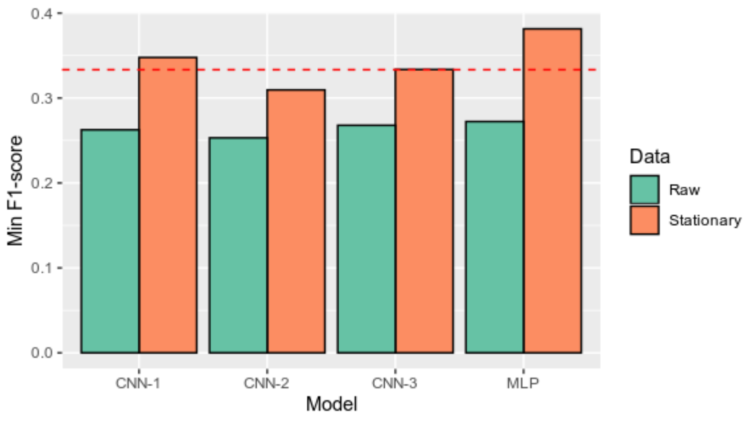

Figure 8: Min F1-score comparison between ANN models with direct encoding

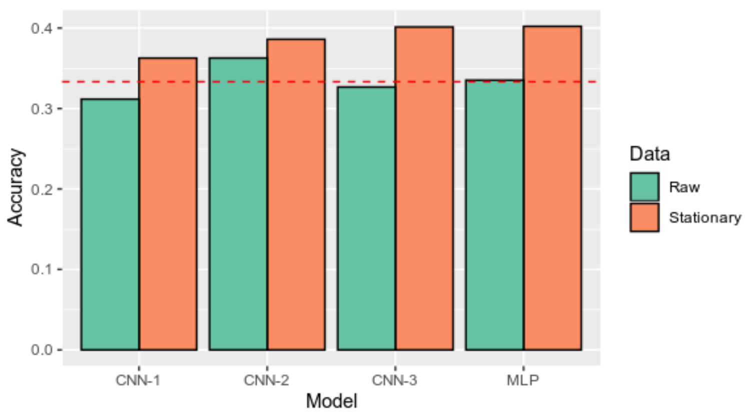

Figure 9: Accuracy comparison between ANN models with direct encoding

### 5.2.2 Experiment 2

This experimented was conducted to check the influence of the size of the training set and the number of $d$ time steps we take in account in the direct encoding representation. We trained an MLP with one hidden layer, since we also wanted to get a benchmark for future CNNs. We decided to use a hidden layer with 512 units because we were to avoid the possibility that a too simple MLP could not learn features of a big input.

We set the maximum number of epochs to 25 and we split the data as follow:

- The training set consisted of the first 10, 60 or 100 consecutive days.
- The validation set consisted of the following 10 consecutive days.
- The test set consisted of the following 10 consecutive days.

The results of figures 10 and 11 show two main conclusions. The first conclusion is that MLP performance improves with more training data. However, after a certain point, the influence is very small. This conclusion was expected, unlike the other conclusion. We observe that increasing the size of the entrance worsens the performance model. This fact seems to indicate that the LOB dynamics mainly depends on the most recent events, therefore, more historical data at the input can introduce noise into the model.

The experiment consequence are the definition of the benchmark as the MLP with a hidden layer with 512 units, the number of $d$ time steps is set to 12, and the size of the training set is 100 days.

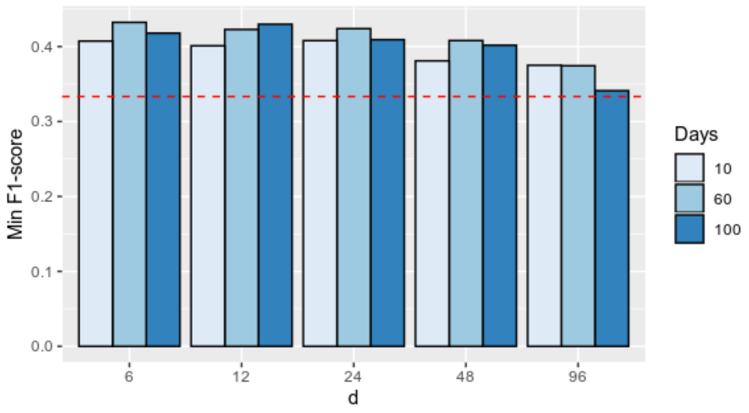

Figure 10: Min F1-score comparison between MLP models with direct encoding

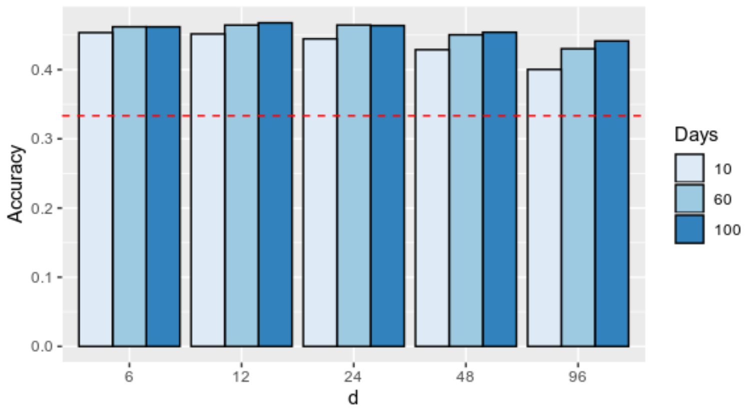

Figure 11: Accuracy comparison between MLP models with direct encoding

### 5.2.3 Experiment 3

The reason behind this experiment was the exploration of other image representations. We tested the GAF representation and the two-channel representation. In both cases, we set the maximum number of epochs to 25 and we split the data as follow:

- The training set consisted of the first 100 consecutive days.
- The validation set consisted of the following 10 consecutive days.
- The test set consisted of the following 10 consecutive days.

The benchmark is the model obtained in the section 5.2.2.

**GAF** The increased data volume involved in this representation imposed a limitation on the number $d$ of snapshots encoded in an image. We restricted $d \in \{4, 8\}$.

We trained three ANN:

- MLP with a hidden layer of 512 neurons. For $d = 4$, it had 164865 trainable parameters. For $d = 8$, it had 656385 trainable parameters.
- A CNN with two sets of one convolutional layer and max pooling. For $d = 4$, it had 36641 trainable parameters. For $d = 8$, it had 110369 trainable parameters. It had the following structure:
  1. Convolutional layer with 48 filters of size $3 \times 3$
  2. Max pooling of $2 \times 2$
  3. Convolutional layer with 48 filters of size $1 \times 1$
  4. Max pooling of $2 \times 2$
  5. Fully-connected layer with 512 neurons
  6. Fully-connected layer with 3 neurons
- A CNN with two sets of two convolutional layer and max pooling. For $d = 4$, it had 59777 trainable parameters. For $d = 8$, it had 133505 trainable parameters. It had the following structure:
  1. Convolutional layer with 48 filters of size $3 \times 3$
  2. Convolutional layer with 48 filters of size $3 \times 3$
  3. Max pooling of $2 \times 2$
  4. Convolutional layer with 48 filters of size $1 \times 1$
  5. Convolutional layer with 48 filters of size $1 \times 1$
  6. Max pooling of $2 \times 2$
  7. Fully-connected layer with 512 neurons
  8. Fully-connected layer with 3 neurons

The results of figure 12 show some difference from the direct encoding, as the stationary time series do not improve the performance. We observed that the performance with $d = 4$ is slightly better than the performance with $d = 8$ when the input came from raw time series. This fact coincides with the conclusions of the section 5.2.2. The performance of the different ANNs is very similar.

Although this representation seems to have potential, due to its high cost of storage, experimentation was stopped. In addition, the small size of the input limits the concatenation of convolutional layers and max pooling layers in a CNN.

**Two-channels** The input image is a three dimensional array of dimensions $r \times d \times 2$. It has two parameters:

- The parameter $r$ is related with the granularity of the prices. If r is very small, most prices collide in the matrix. If r is very large, the matrix contains practically only zeros. We explored the values of $r \in \{4, 8\}$
- The parameter $d$ is related with the memory of the image. We explored the values of $d \in \{4, 8, 16, 64\}$.

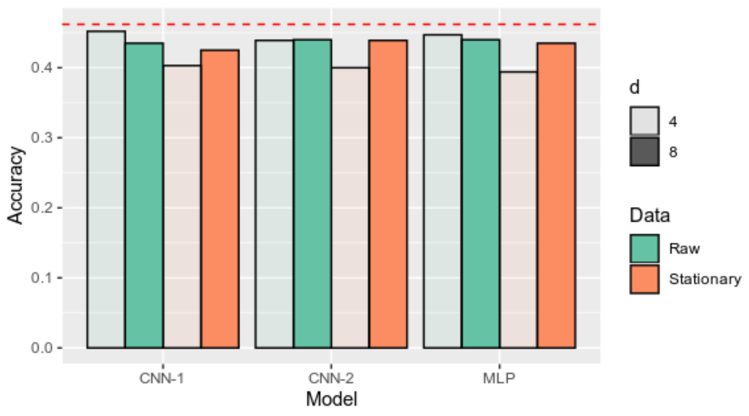

Figure 12: Accuracy comparison between models with GAF encoding

We trained two ANN:

- MLP with a hidden layer of 512 neurons.
- A CNN with two sets of two convolutional layer and max pooling. For $d = 4$, it had 59777 trainable parameters. For $d = 8$, it had 133505 trainable parameters. It had the following structure:
    1. Convolutional layer with 48 filters of size $3 \times 3$
    2. Convolutional layer with 48 filters of size $3 \times 3$
    3. Max pooling of $2 \times 2$
    4. Convolutional layer with 48 filters of size $1 \times 1$
    5. Convolutional layer with 48 filters of size $1 \times 1$
    6. Max pooling of $2 \times 2$
    7. Fully-connected layer with 512 neurons
    8. Fully-connected layer with 3 neurons

The results of figure 13 only show small differences depending on the choice of the parameter or model value. MLP performance worsens as the amount of historical data in the image increases, but CNN performance is almost constant. One possible explanation, which follows the conclusions of section 5.2.2, is that data from older past snapshots has little impact on future changes and increases noise. Because CNN can detect features, it may focus on information from the latest snapshots.

This image representation needs to tune one parameter more than the other representations to find the most suitable pair $(r, d)$. Since its results are not better than in the direct encoding, we decided to refuse this representation.

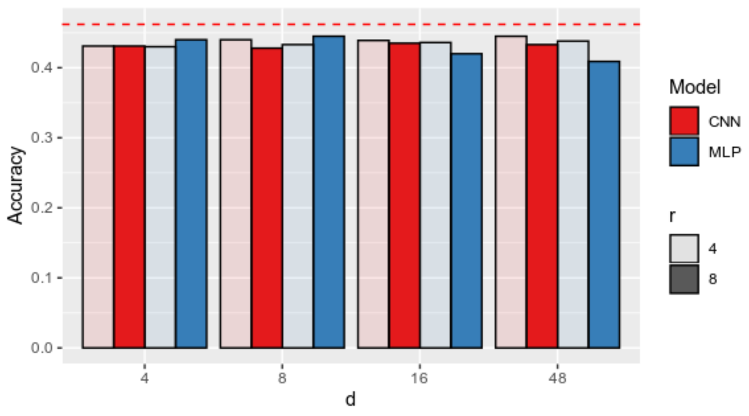

Figure 13: Accuracy comparison between models with two-channels encoding

### 5.2.4 Experiment 4

This experiment was conducted to explore more complex CNN than in section 5.2.1. For this experiment, we set $d = 12$. The benchmark was the MLP from section 5.2.2.

We set the maximum number of epochs to 25 and we split the data as follow:

- The training set consisted of the first 100 consecutive days.
- The validation set consisted of the following 10 consecutive days.
- The test set consisted of the following 10 consecutive days.

We defined a CNN as CNN-$s$-$c$ if its structure consist in $s$ blocks of $c$ convolutional layers and one max pooling layer, with the rules explained in section . We trained a CNN-1-2, CNN-2-2, CNN-2-3, CNN-3-2 and CNN-3-3. The number of filters in the first convolutional layer is 32.

In addition, we designed a version of CNN-2-2 and CNN-2-3 such that they had shortcut connection as in figure 5. CNN-3-2 and CNN-3-3 also have a trained version that had shortcut connection. However, in these models, the shortcut skips two convolutional layers instead of one. In conclusion, nine CNN were trained. The legend *ResNet* in the figures 14 and 15 represents if the model has or not shortcut connection: $Y$ is the affirmative case, $N$ is the negative case.

The results of figures 14 and 15 confirm that the shortcut connection have a good impact in the performance of the CNN, as their accuracy is always improved, specially in CNN-2-2 case. Although CNN-2-2 with shortcuts is the best model and it has a slightly better performance than the benchmark, all four CNNs with shortcuts and the MLP obtain very similar results.

One possible explanation is the nature of the problem itself. The maximum accuracy achievable with these labels by ANNs is significantly less than 1 and our current results are close to that accuracy. This hypothesis may be supported by the fact that all the performance are close to the best one. Another explanation is that our models need more expressiveness to learn features of the data.

Regardless of the reason, this accuracy is insufficient to define a trading strategy without handcrafted features. Because of that, in the section 4.7 we defined two new labels that were motivated to detect when to trade. Thus, the accuracy does not need to be high, but we must reduce false positives.

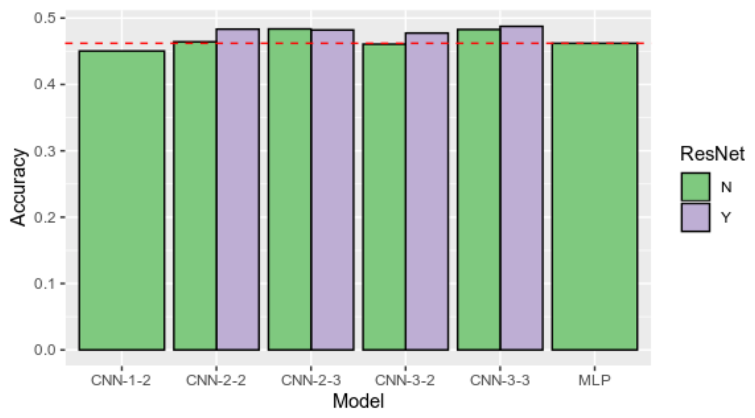

Figure 14: Accuracy comparison between MLP and CNNs with direct encoding

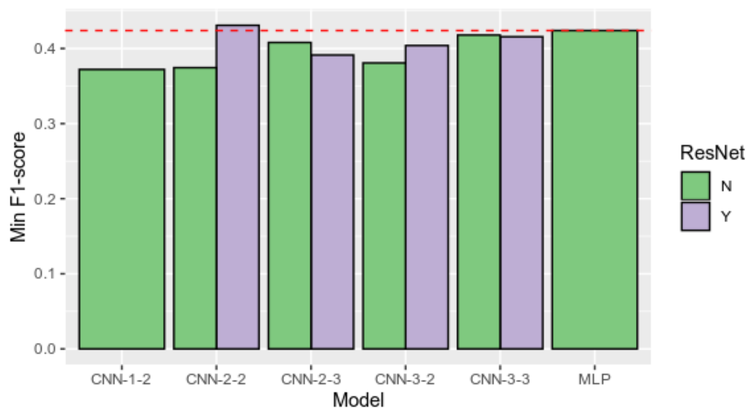

Figure 15: Min F1-score comparison between MLP and CNNs with direct encoding

## 5.3 Proposed labels performance of MLP and different CNN

The motivation of the possibility that the mid price based labels are not the most suitable for LOB forecasting, we conducted experiments to observe the neural network performance in the prediction of the label $l_1$ and $l_2$, based on the difference between the current best ask price and the best bid prices in an horizon h. The prediction is a number between 0 an 1, so we can define a threshold $\pi$ such that if the prediction is greater than $\rho$, the forecast is the class 1. On the contrary, if the prediction is less than $\pi$, the forecast is the class 0.

For these experiments, we set the maximum number of epochs to 25 and we split the data as follow:

- The training set consisted of the first 80 consecutive days.
- The validation set consisted of the following 10 consecutive days.
- The test set consisted of the following 10 consecutive days.

Analogously, we also used early stopping to avoid the overfitting and the data was split in training set, validation test and test set. The validation set is also used to choose $\pi$ as the minimum threshold that maximizes the trading score in the validation set.

We decided to set the horizon to $h = 20$. Due to the analysis of the previous experiments, we used the direct encoding with $d = 12$.

### 5.3.1 Experiment 1

The first experiment was realized to have insights of the labels $l_1$ an $l_2$ and their thresholds. For the experiments, we trained a MLP with a hidden layer with 512 units.

We trained one model for a bunch threshold of each label.

- For $l_1$, we trained models for the thresholds $\tau_1 \in \{0, \frac{1}{20}, \frac{2}{20}, \frac{3}{20}, \frac{4}{20}, \frac{6}{20}, \frac{9}{20}, \frac{12}{20}\}$.
- For $l_2$, we trained models for the thresholds $\tau_2 \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.2\}$.

For each trained model, we found the combination of a threshold $\pi$ and a parameter $\rho$ such that maximizes the trading score of the validation set. The possible values for $\rho$ are $\{0.2 \cdot k | 1 \leq k \leq 10\}$. The loss function is the binary cross-entropy.

The figure 16 show the best five trading score for each label on the test set. We observe that $\rho = 0.6$ is the most common for the parameter. We expected that the label $l_2$ would have a better performance, because is based on the maximum difference between the current best ask price and the future best bid prices. However, the best performance was achieved thanks to the label $l_1$ when $\tau_1 \in \{\frac{4}{20}, \frac{6}{20}\}$, as the figure 17 expresses.

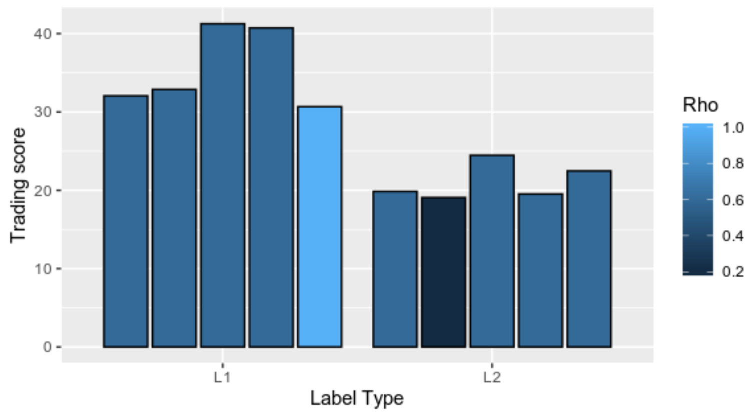

Figure 16: Trading score comparison between two labels

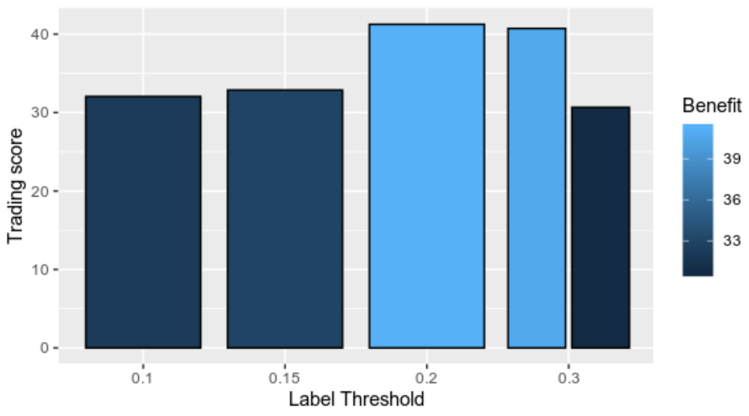

Figure 17: Trading score comparison between the best label

### 5.3.2 Experiment 2

Since our data is non balance, following the work in [Lin+17], we explored the focal loss. We conducted experiments to choose the parameters $\alpha, \gamma$ by training a MLP with a hidden layer with 512 units. The possible values for $\alpha$ were $\{0.25, 0.5, 0.75\}$, and the possible values for $\gamma$ were $\{0.1, 0.25, 0.5, 2, 5\}$. After the previous experiment, the $\rho$ value was fixed to 0.6.

We trained models for the labels $l_1$ and the thresholds $\tau_1 \in \{0, \frac{2}{20}, \frac{4}{20}, \frac{6}{20}\}$. For each trained model, we found the threshold $\pi$ that maximizes the trading score of the validation set.

For all thresholds, the best combination $(\alpha, \gamma)$ was $(0.5, 0.5)$. For each threshold $\tau_1$ tested, the figure 18 shows the comparison of the trading score between the best results using focal loss and its analogous using cross-entropy as loss function. From the figure, we observe that for $\tau_1 \in \{0, 2\}$, the focal loss may seem a good idea, but the performance worsen with $\tau_1 \in \{4, 6\}$.

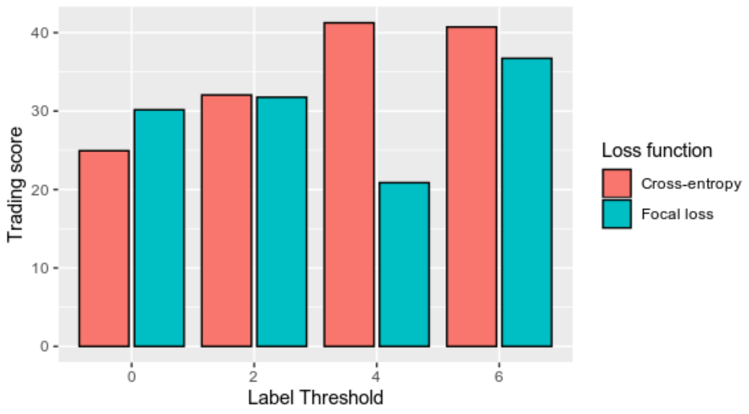

Figure 18: Trading score comparison between focal loss and cross-entropy

### 5.3.3 Experiment 3

The final experiment was conducted to compare the MLP against the CNN-2-2 with shortcuts, since in section 5.2.4, this model outperform the MLP. We trained four CNNs, since we wanted to test the labels $l_1$ and the thresholds $\tau_1 \in \{\frac{4}{20}, \frac{6}{20}\}$ and both loss functions, i.e. binary cross-entropy and focal loss with $(\alpha, \gamma) = (0.5, 0.5)$. Similarly to the previous experiment, for each trained model, we found the threshold $\pi$ that maximizes the trading score of the validation set.

The results are summarised in the figure 19. Although the CNN model obtained a positive trading score, its performance is worse than the performance of MLP models. Due to the higher time cost of training compared to MLP, the number of experiments performed is significantly less. The last observation is that the focal loss continues to worsen the results for $\tau_1 \in \{4, 6\}$.

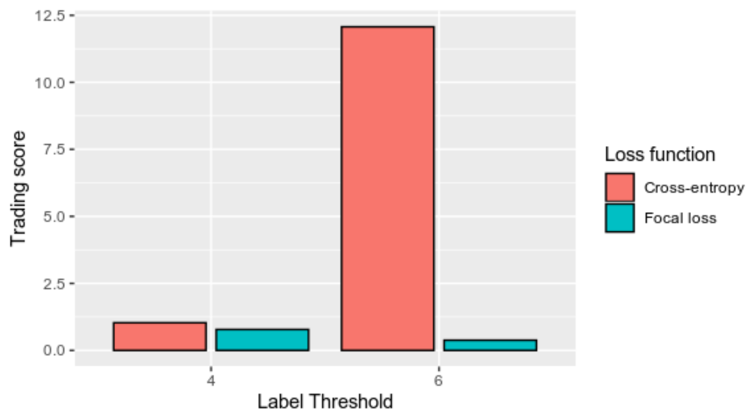

Figure 19: Trading score comparison between focal loss and cross-entropy

In conclusion, it was possible to trade with positive gains directly from the predictions of an ANN. However, we have not been able to improve the results of the MLP with CNNs.

# 6 Conclusion

We started with a traditional time series analysis that failed to accurately capture the nonlinearity nature of LOB, followed by an ANN study applied to a classification problem.

At first, we designed CNN to try to predict the direction of the mid price, which is a common approach in the literature. Although the experiments showed that CNNs were able to capture LOB nonlinearities and improve the results achieved with MLP, they did not achieve high accuracies. Different methodologies have been applied to correct this problem, as testing different architectures and LOB representation, but they have not offered good solutions.

At last, we have introduced new labels with the motivation to predict when the trade reports benefits. Although we have achieved the objective of developing CNN models that obtain positive results, these are inferior to those obtained with MLP.

Further development in the project may include further experimentation with CNN in predicting the new labels. After that, a possible approach would be to try to eliminate the labeling process by using Reinforcement learning algorithms. Since in the context of HFT is easy to define a reward, for example, the obtained benefit, these algorithms seem a good option.

# 7 Annex: Simple models

In this section, we expand the description of the simple models. They are going to be defined for time series $y$, where $y_1 \cdots y_T$ is the historical data and $\hat{y}_{T+h|T}$ is the estimate of $y_{T+h}$ based on the historical data.

## 7.1 ARIMA

The mathematical formulation of the ARIMA(p,d,q) model using lag polynomial is:
$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right)(1-L)^d y_t = \left(1 - \sum_{i=1}^{p} \theta_i L^i\right)$

where $p$, $d$, $q$ are nonnegative integers and refer to the order of the autoregressive, integrated and moving average parts of the model. The *lag operator* is define as $Ly_t = y_{t-1}$

In the case of $d = 1$, the model can be rewritten as:

$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdot + \theta_q \varepsilon_{t-q} + \varepsilon_t$ where $y'_t$ is the differenced series.

If $y_t$ is differenced once, the result series is $y'_t = y_t - y_{t-1}$. We can difference a time series more than once. For instance, the twice-differenced series is $y''_t = y'_t - y'_{t-1} = y_t - y_{t-1} - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2}$

## 7.2 Naive

A naive forecast is optimal when data follow a random walk. Since the random walk is a common model for the evolution of stock prices, it is a reasonable benchmark. The mathematical formulation is:

$$\hat{y}_{T+h|T} = y_T$$

This model is equivalent to an ARIMA(0,1,0).

## 7.3 Exponential smoothing

One can choose whether to have trend and seasonal terms and also one can choose whether they are additive or multiplicative. This can be expressed compactly with the triplet (E,T,S) refers to the three components: error, trend and seasonality. This notation helps in specifying how the components are specified: N, A, M refer to none, additive and multiplicative respectively.

For example, the simplest model is ETS(A,N,N) and is also called simple exponential smoothing. This forecast has the form of:

$$\hat{y}_{T+1|T} = \alpha y_T + (1-\alpha)\hat{y}_{T|T-1}$$

.

All forecast take the same value $\hat{y}_{T+h|T} = \hat{y}_{T+1|T}$

The linear exponential smoothing models are all special cases of ARIMA models (ETS(A,N,N) corresponds to ARIMA(0,1,1) and ETS(A,A,N) corresponds to ARIMA(0,2,2)). However, the non-linear exponential smoothing models have no equivalent ARIMA counterparts. On the other hand, there are also many ARIMA models that have no exponential smoothing counterparts. In particular, all ETS models are non-stationary, while some ARIMA models are stationary.

# 8 Annex: Custom loss function

The categorical cross-entropy measures the difference between the distributions of the real and predicted labels. We explored the idea of using the confusion matrix to penalize errors more than others -for example, the penalty of predicting downward mid price direction instead of upward should be greater than the penalty of predicting no significant change and being wrong. The reason comes from the financial point of view, since it is evident that there are misprediction that could cause more losses than others.

Therefore, we explored the idea of a custom loss function which tries to accomplish the task of scoring the confusion matrix. Since the loss functions must be differential, we approximate as:

$$Q(\mathbf{y}, \hat{\mathbf{y}}) = \mathbf{y}^t \hat{\mathbf{y}}$$

where $\mathbf{y}, \hat{\mathbf{y}} \in \mathrm{R}^{N \times C}$ are bunch of $N$ row vector and $C$ numbers of classes. The vector $\mathbf{y}_i$ represents the target prediction and $\hat{\mathbf{y}}_i$ the model outcome for the same input. To score $Q$, we add the elements of the matrix resulting from the element-wise product of the matrix $Q$ and the weight matrix $\boldsymbol{\theta}$.

$$q(Q; \boldsymbol{\theta}) = I_C \cdot (\boldsymbol{\theta} \odot Q) \cdot I_C^t$$

where $I_C$ is the unity row vector $(1, \dots, 1)$ of dimension $1 \times C$.

We observed that

$$q(Q(\mathbf{y}, \hat{\mathbf{y}}); \boldsymbol{\theta}) = \sum_{i=1}^{N} q(Q(\mathbf{y}_i, \hat{\mathbf{y}}_i); \boldsymbol{\theta})$$

We restricted ourselves to a model prediction and its target value to study and define the function. In this way, the function can be rewritten as:

$$\mathcal{L}(y, \hat{y}; \boldsymbol{\theta}) = -\sum_{i=1}^{C} \sum_{j=1}^{C} \boldsymbol{\theta}_{ij} \cdot y_i \cdot \hat{y}_j$$

It has a minus sign so that minimizing the loss function is maximizing the matrix score.

## 8.1 Confunsion matrix

It is a multiclass loss, so the activation function of the output layer is *softmax*. Let be $z$ the output of the model before applying the activation function and $p$ the output of the model after applying the activation function. Then,

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

Since the weights are updated by a stochastic gradient descent algorithm, the update depends on the gradient of the loss function respect to the weight. In general, this gradient is calculated by the chain rule, so it depends heavily on the gradient of the loss function respect to the output of the model.

$$\begin{aligned}
\frac{\partial \mathcal{L}(y,p)}{\partial z_k} &= -\sum_{i=1}^{C}\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}\cdot y_i\frac{\partial p_j}{\partial z_k} = \\
&= -\sum_{i=1}^{C}\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}\cdot y_i p_j(\delta_{jk}-p_k) = \\
&= -\sum_{i=1}^{C}\left(-\sum_{\substack{j=1\\ j\neq k}}^{C}(\boldsymbol{\theta}_{ij}y_i p_j p_k) + \boldsymbol{\theta}_{ik}\cdot y_i p_k(1-p_k)\right) = \\
&= -p_k\sum_{i=1}^{C} y_i\left(\boldsymbol{\theta}_{ik}-\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}p_j\right) = -p_k\sum_{i=1}^{C}\left(y_i\sum_{j=1}^{C}(\boldsymbol{\theta}_{ik}-\boldsymbol{\theta}_{ij})p_j\right)
\end{aligned}$$

When $y = p$, the derivative of loss function is zero. We want this since the red neuronal does not need any update. However, when $y = (1,0,0)$, then $p = (0, p_2, p_3)$ is also a point that cancel the derivative. Because of that, we discard this function, since it has bad properties.

An example is:

$$\theta = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix} \implies \frac{\partial \mathcal{L}(y,p)}{\partial z_k} = -p_k\sum_{\substack{j=1\\ j\neq k}}^{C} y_k p_j = -p_k(1-p_k)y_k$$

where the penalization for the penalty for a very incorrect prediction is the same as for a very correct one.

## 8.2 Logarithm

We add a function $f$ to the loss function to try to get another one with better properties.

$$\mathcal{L}(y,\hat{y};\boldsymbol{\theta},f) = -\sum_{i=1}^{C}\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}\cdot y_i\cdot f(\hat{y}_j)$$

Then,

$$\begin{aligned}
\frac{\partial \mathcal{L}(y,p)}{\partial z_k} &= -\sum_{i=1}^{C}\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}\cdot y_i f'(p_j)\frac{\partial p_j}{\partial z_k} = \\
&= -p_k\sum_{i=1}^{C} y_i\left(\boldsymbol{\theta}_{ik}f'(p_k)-\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}f'(p_j)p_j\right) = \\
&= -p_k\sum_{i=1}^{C}\boldsymbol{\theta}_{ik}y_i f'(p_k) + p_k\sum_{i=1}^{C}\sum_{j=1}^{C}\boldsymbol{\theta}_{ij}y_i f'(p_j)p_j
\end{aligned}$$

To avoid unwanted zeros, necessarily if $y_k = 1$, then $\lim_{p_k \to 0^+} \frac{\partial \mathcal{L}(y,p)}{\partial z_k} = c$, where $c$ is a non zero constant. The condition has to be satisfied for every $p_j$, where $j \neq k$. Since $f$ is not the constant function:

$$\lim_{p_k \to 0^+} p_k \sum_{i=1}^{C} \sum_{j=1}^{C} \boldsymbol{\theta}_{ij} y_i f'(p_j) p_j = 0$$

Therefore,

$$\lim_{p_k \to 0^+} \frac{\partial \mathcal{L}(y,p)}{\partial z_k} = \lim_{p_k \to 0^+} -p_k \sum_{i=1}^{C} \boldsymbol{\theta}_{ik} y_i f'(p_k) = -\boldsymbol{\theta}_{kk} \lim_{p_k \to 0^+} p_k f'(p_k) = c \implies \lim_{x \to 0^+} x f'(x) = \lim_{x \to 0^+} \frac{f'(x)}{\frac{1}{x}} = c'$$

We can conclude that $f'(x) \sim \frac{1}{x}$ as $x \to 0^+$. A natural option for $f'(x)$ is $\frac{1}{x}$. Because of that, we set $f(x) = \log(x)$:

$$\begin{aligned} \frac{\partial \mathcal{L}(y,p)}{\partial z_k} &= -\sum_{i=1}^{C} y_i \left( \boldsymbol{\theta}_{ik} - \sum_{j=1}^{C} \boldsymbol{\theta}_{ij} p_k p_j \right) = \\ &= -\boldsymbol{\theta}_{tk} + p_k \sum_{j=1}^{C} \boldsymbol{\theta}_{tj} p_j \text{ where } y_i = \delta_{it} \end{aligned}$$

Assuming that $k \neq t$ and $p_k = 0$, then $\frac{\partial \mathcal{L}(y,p)}{\partial z_k} = -\boldsymbol{\theta}_{tk}$

In conclusion, the loss function should have the form:

$$\mathcal{L}(y, \hat{y}; \boldsymbol{\theta}, \log) = -\sum_{i=1}^{C} \boldsymbol{\theta}_{ii} \cdot y_i \cdot \log(\hat{y}_i)$$

We note that this function is a weighted categorical cross-entropy.

# References

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning, Second Edition (Adaptive Computation and Machine Learning)*. 2nd ed. Adaptive Computation and Machine Learning. The MIT Press, 2010. ISBN: 026201243X,9780262012430. URL: `http://gen.lib.rus.ec/book/index.php?md5=78e4afc293ef14dfe964fc156cd079c0`.

[BKK18] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *CoRR* abs/1803.01271 (2018). arXiv: `1803.01271`. URL: `http://arxiv.org/abs/1803.01271`.

[Bas+18] Amitabh Basu et al. "Convergence guarantees for RMSProp and ADAM in non-convex optimization and their comparison to Nesterov acceleration on autoencoders". In: *CoRR* abs/1807.06766 (2018). arXiv: `1807.06766`. URL: `http://arxiv.org/abs/1807.06766`.

[Bis07] Christopher M. Bishop. *Natural networks for pattern recognition*. Oxford Univ. Press, 2007.

[Hau14] Nikolaus Hautsch. *Econometrics of financial high-frequency data*. Springer, 2014.

[Hay16] Simon Haykin. *Neural networks and learning machines*. 3rd ed. Delhi Chennai: Pearson, 2016.

[Hyn08] Rob Hyndman. *Forecasting with exponential smoothing: the state space approach*. Springer, 2008.

[HA18] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. 3rd ed. OTexts, 2018.

[Lin+17] Tsung-Yi Lin et al. "Focal Loss for Dense Object Detection". In: *CoRR* abs/1708.02002 (2017). arXiv: `1708.02002`. URL: `http://arxiv.org/abs/1708.02002`.

[Lu+17] Zhou Lu et al. "The Expressive Power of Neural Networks: A View from the Width". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 6231–6239. URL: `http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf`.

[Nou+18] Paraskevi Nousi et al. "Machine Learning for Forecasting Mid Price Movement using Limit Order Book Data". In: *CoRR* abs/1809.07861 (2018). arXiv: `1809.07861`. URL: `http://arxiv.org/abs/1809.07861`.

[Nta+17] Adamantios Ntakaris et al. "Benchmark Dataset for Mid-Price Prediction of Limit Order Book data". In: *CoRR* abs/1705.03233 (2017). arXiv: `1705.03233`. URL: `http://arxiv.org/abs/1705.03233`.

[Sil+17] Ivan Nunes da Silva et al. *Artificial Neural Networks*. Springer International Publishing, 2017. DOI: `10.1007/978-3-319-43162-8`. URL: `https://doi.org/10.1007/978-3-319-43162-8`.

[SZ14a] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: `1409.1556 [cs.CV]`.

[SZ14b] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: `1409.1556 [cs.CV]`.

[Tra+20] Dat Thanh Tran et al. *Data Normalization for Bilinear Structures in High-Frequency Financial Time-series*. 2020. arXiv: `2003.00598 [cs.CE]`.

[Tsa16] Ruey S. Tsay. *Analysis of financial time series*. Wiley India, 2016.

[WZ15] Zhaodong Wang and Weian Zheng. *High-frequency trading and probability theory*. World Scientific, 2015.

[WO15] Zhiguang Wang and Tim Oates. *Imaging Time-Series to Improve Classification and Imputation*. 2015. arXiv: `1506.00327 [cs.LG]`.

[YD19] Omolbanin Yazdanbakhsh and Scott Dick. "Multivariate Time Series Classification using Dilated Convolutional Neural Network". In: *CoRR* abs/1905.01697 (2019). arXiv: `1905.01697`. URL: `http://arxiv.org/abs/1905.01697`.

[ZZR19] Zihao Zhang, Stefan Zohren, and Stephen Roberts. "DeepLOB: Deep Convolutional Neural Networks for Limit Order Books". In: *IEEE Transactions on Signal Processing* 67.11 (June 2019), pp. 3001–3012. ISSN: 1941-0476. DOI: `10.1109/tsp.2019.2907260`. URL: `http://dx.doi.org/10.1109/TSP.2019.2907260`.

[Zhe+15] Yi Zheng et al. "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification". In: *Frontiers of Computer Science* 10.1 (Aug. 2015), pp. 96–112. DOI: `10.1007/s11704-015-4478-2`.