



## Deep learning in exchange markets<sup>☆</sup>

Rui Gonçalves<sup>a</sup>, Vitor Miguel Ribeiro<sup>c,\*</sup>, Fernando Lobo Pereira<sup>a</sup>, Ana Paula Rocha<sup>b</sup>

<sup>a</sup> Department of Electrical and Computer Engineering, School of Engineering, Porto University (FEUP), Portugal

<sup>b</sup> Department of Informatics Engineering, School of Engineering, Porto University (FEUP), Portugal

<sup>c</sup> Department of Economics, Aveiro University (DEGEIT), Portugal



### ARTICLE INFO

#### Article history:

Received 31 March 2018

Revised 15 May 2019

Accepted 28 May 2019

Available online 29 May 2019

#### JEL classification:

G17

D10

L10

#### Keywords:

Deep learning  
Betting exchange  
Market depth  
Classification

### ABSTRACT

We present the implementation of a short-term forecasting system of price movements in exchange markets using market depth data and a systematic procedure to enable a fully automated trading system. Three types of Deep Learning (DL) Neural Network (NN) methodologies are trained and tested: Deep NN Classifier (DNNC), Long Short-Term Memory (LSTM) and Convolutional NN (CNN). Although the LSTM is more suitable for multivariate time series analysis from a theoretical point of view, test results indicate that the CNN has on average the best predictive power in the case study under analysis, which is the UK to Win Horse Racing market during pre-live stage in the world's most relevant betting exchange. Implications from the generalized use of automated trading systems in betting exchange markets are discussed.

© 2019 Elsevier B.V. All rights reserved.

### 1. Introduction

The increasing amount of data reveals that the Big Data era is here to stay and constitutes a new form of strategic behavior and business interaction. Data is currently considered one of the most valuable intangible assets in the world. The domain of data analytical techniques is a key step not only to facilitate the transformation and growth of firms but also to boost the level of digital literacy. Goodfellow et al. (2016) recognize that the use of Deep Learning (DL) constitutes an enabler of disruptive change for businesses due to its power of association, regression, classification and clustering. Machine learning incorporates a vast array of algorithmic implementations, which not all of them can be classified as DL. Indeed, the later only corresponds to a subset of the former field of research.

Historically emerging from cognitive and information theories, DL aims at imitating the learning process of human neurons and creates complex interconnected neuronal structures sim-

ilar to human synapses. Hence, DL consists of the application of multi-neuron, multi-layer Neural Networks (NN) to perform learning tasks such as regression, classification, clustering or encoding/decoding. The ability for a NN to be used in a wide variety of data and learn indiscriminately implies that the DL approach can be applied to a considerable number of case studies rather than requiring the development of a structure for each new analysis. Varian (2014) recognizes the relevance of DL NN architectures for the economics field. Proficiency with data mining, data visualization tools and artificial intelligence rank as one of the most important skills in determining business success, thus, any effort to educate stakeholders is clearly advised.

This study presents a framework for short-term forecasting of price movements in exchange markets and, therefore, the main core relies on time series forecasting. It is focused on modeling predictors of future values of a time series based on past observations. The relationship between past and future observations in the domain of financial markets is stochastic or non-deterministic, which implies that the conditional probability distribution of a matrix of inputs (X) as a function of past observations is generically given by:

$$P(X_{t+d}|X_t, X_{t-1}, \dots) = f(X_t, X_{t-1}, \dots)$$

DL NN models have built-in properties that make them suitable for multivariate time series analysis. Firstly, in their basis, they are robust to noise in input data and can support learning and prediction

<sup>☆</sup> The authors appreciate the valuable suggestions provided by the Associate Editor Scott Savage and anonymous referees, which significantly improved the quality of the study. We acknowledge financial support from the project STRIDE - NORTE-01-0145-FEDER-000033.

\* Corresponding author.

E-mail addresses: [rjgg@fe.up.pt](mailto:rjgg@fe.up.pt) (R. Gonçalves), [vmsribeiro@ua.pt](mailto:vmsribeiro@ua.pt) (V.M. Ribeiro), [flp@fe.up.pt](mailto:flp@fe.up.pt) (F.L. Pereira), [arocha@fe.up.pt](mailto:arocha@fe.up.pt) (A.P. Rocha).

in the presence of missing values (Dixon et al., 2015). Secondly, they do not make strong assumptions about the mapping function and learn either linear or non-linear relationships (Dorffner, 1996). This implies that they add the capability to learn non-linear relationships with arbitrarily defined, though fixed, number of inputs and outputs (Huck, 2009; 2010). This is extremely important because most real-life events are characterized by complex relationships. Thirdly, they have generalization power due to the recognition of unobserved relationships in data after learning from a set of inputs. Fourthly, they are not excessively rigid on the treatment of input data (e.g., forcing the persistence of a certain distribution). Fifthly, they deal better with heteroskedasticity due to their ability to learn hidden relationships in data without the imposition of additional constraints. Finally, in the case of recurrent neural networks (RNNs), they learn temporal dependence from context. The study of Hochreiter and Schmidhuber (1997) is the seminal contribution responsible for the introduction of Long Short-Term Memory (LSTM), which is a particular type of RNN that has played a key role in recent DL advances due to its learning ability in long run dependencies. From a theoretical point of view, the LSTM learns adequately long-term dependencies between time steps of sequence data and, therefore, it is frequently considered the benchmarking DL NN model for multivariate time series analysis (Fischer and Krauss, 2018).

The DL approach contemplates a meaningful set of application fields (Hatcher and Yu, 2018). Financial exchange markets have only recently been subject to the ability of DL NN models to learn stochastic data. In general, researchers intend to implement predictive mechanisms through DL NN architectures in order to recognize trends and detect anomalous behavior. Ding et al. (2015) analyze stock exchange market price predictions through the implementation of a deep NN to learn event embedding and a Convolutional NN (CNN) for short-, medium- and long-term analysis. Their model improves accuracy and profit relatively to baseline NN methods, especially for firms with a lower number of available news. Heaton et al. (2016) consider a DL auto-encoding technique that is based on the principal component analysis (PCA) for high-dimensionality data reduction, which allows feature extraction. This way, they define a smart index (i.e. a fit approximation of a subset of stocks to a single index). Korczak and Hemes (2017) show that, compared to simple NN multilayer perceptron (MLP), a CNN in the H2O algorithmic trading framework significantly increases the average rate of return per transaction in the FOREX exchange market. Hu et al. (2018) consider a Hybrid Attention Network (HAN) to predict stock exchange market trends based on the report of news. Their natural language processing (NLP) framework internalizes attention to values of temporal news vectors and uses LSTM for sequential modeling prediction. Their training mechanism increases accuracy and outperforms competing methods in simulation. Zhao et al. (2017) ensembles different models extracted with a cross validation process (i.e. division of the training dataset into multiple subsets) to forecast the West Texas Intermediate crude oil spot price using stacked auto-encoders (SAE). Fischer and Krauss (2018) consider LSTM networks for predicting price movements for the constituent stocks of the S&P 500. In their case study, LSTM networks have a better predictive power than memory-free classification networks. They also reveal returns close to zero after 2009 due to the low exposure of the trained model to systemic risks.

The main objective of this study is to implement a short-term price movement forecast system in betting exchange markets (i.e. classify changes in odds). This type of exchange shares common grounds with financial exchange markets, namely in terms of raw data format and framework interaction, as clarified in Section 2. We developed agents that execute trades of bets on the United Kingdom (UK) to Win Horse Racing market of the Betfair betting

exchange. The period of actuation is the 10 min time window before the start of a race. During this period, the odds (i.e. prices) of bets are subject to speculation. Our agents try to establish a profit by buying and selling bets at different prices before the beginning of a race. Three types of DL NN architectures are trained, tested and compared against each other to find the one that discloses the best price movement prediction: Deep NN Classifier (DNNC), LSTM and CNN.

Two main results are provided. First, after exposing the implementation procedure of the three distinct DL NN architectures, we conclude that the CNN ensures the highest level of accuracy. This implies that, although the LSTM is more suitable for multivariate time series analysis from a theoretical point of view, the CNN outperforms the LSTM. Therefore, in our case study, the use of models with memory cells (e.g., LSTM) can be negligible since the time series under analysis change context periodically (i.e. each 10 min pre-race event constitutes a different context). Second, validation results show that all DL NN models ensure a positive, though low, profit and loss (PL) at the end of a simulation conducted during 30 days.

The remaining of the article is organized as follows. Section 2 presents the case study. Section 3 provides an overview of the different DL NN architectures considered in this study. Section 4 describes the methodology. Section 5 exposes the results. Conclusions are summarized in Section 6.

## 2. Case study

Our goal is to accurately estimate changes in odds to buy and sell bets and try to guarantee a profit. We focus on the Betfair betting exchange, which is the largest of its kind in the world, with the majority of customers based in the UK (Brown and Yang, 2017). We act on the pre-race market, particularly in the last 10 min of trading prior to the start of a race. In the majority of modeling problems, unexpected external factors can override assumptions in which the predictive system relies on. However, under this time period, the market under analysis corresponds to a closed loop system where only internal market data have the ability to induce price fluctuations, so that the predictive power is exclusively dependent on market data itself. As such, this study is exclusively concerned with purely speculative markets.

### 2.1. Raw data collection

Raw data reveal that the 10 min time window before the start of a race is when the market becomes more active. Fig. 1 exposes the average value of trading volume, liquidity (i.e. sum of amounts waiting to be matched at the bid and ask price) and volatility (i.e. number of ticks<sup>1</sup> variation in absolute value per minute) for the complete sample of observed races, considering all runners involved in a given race. From the 10th minute before each race starts until the effective start of each race, the average trading volume increases about 4.2 times, the average liquidity increases approximately 3.4 times and the average volatility increases about 1.6 times. Hence, from a dynamic point of view, we observe that all variables increase as we approach the beginning of a race.

Raw data were collected directly from Betfair servers in real time at a twice per second rate from the 1st September 2014 to the 29th August 2016. For the sake of brevity, summary statistics with respect to volume, liquidity and volatility at the race level on a per minute basis during the 10 min time window before the start of each race are provided in Table A.1 of the Appendix. Globally, we recorded a personalized database related to the UK to Win Horse

<sup>1</sup> The unit of measure of a price change is designated by tick.

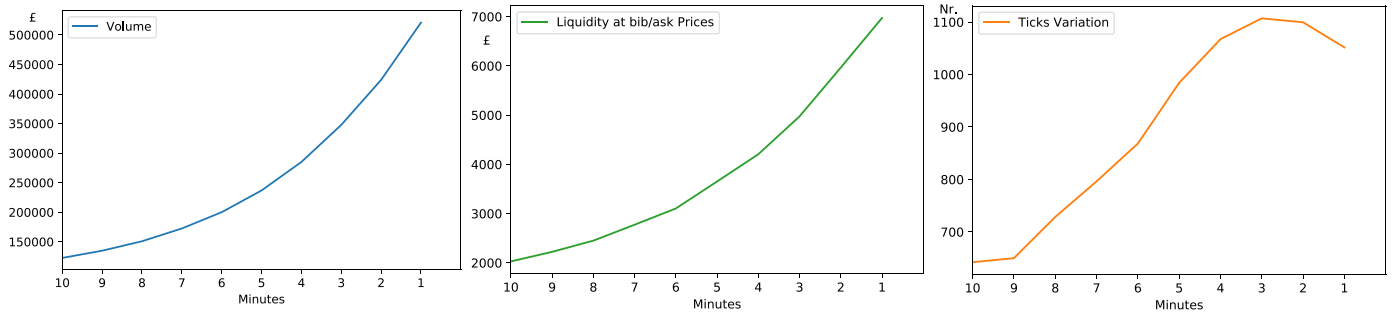


Fig. 1. Average trading volume, average liquidity at the bid and ask price, and average number of ticks variation in absolute value per minute.

Table 1  
Snapshot in time of market depth, raw data frame information.

Buy	Bid	Price	Ask	Sell	Volume
		∴			
		5,1			20
		5,0	250		93
		4,9			68
		4,8	263		24
		4,7	148	10,00	70
		4,6	349	5,00	76
	8	4,5			217
	2	4,4			23
10,00	10	4,3			4
	448	4,2			
	398	4,1			
	335	4,0			
		∴			

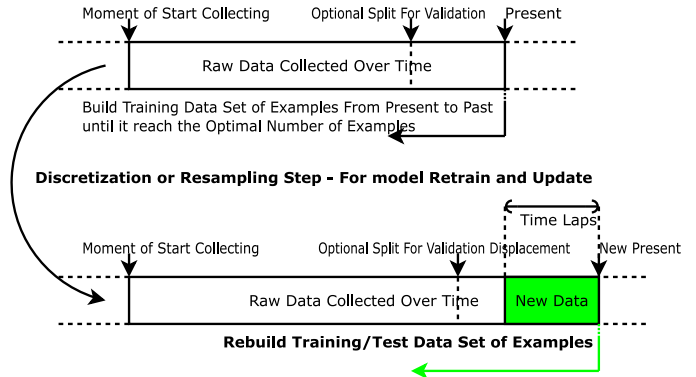


Fig. 2. New information added to the raw dataset and re-training iteration.

Racing market characterized by 15 to 30 daily races with 3 to 25 competing horses per race.

The methodology employed in this study is only applied to exchanges that provide market depth data, also known as level II market data<sup>2</sup> To illustrate the format of a market depth view, Table 1 presents a frame of information stopped in time exemplifying a ladder view. The Price column describes the possible prices. The buy and sell amounts are listed in the Bid and Ask columns. The Buy and Sell columns represent orders of an agent that are waiting to be matched. When the buy (i.e. back) and sell (i.e. lay) orders reach the same price level, there is a matched amount transaction. The total amount transacted at each price is listed in the Volume column. The yellow cell shows the last matched price. All these information are referred throughout the manuscript as raw data frame (RDF) and reflect that we have collected data at the horse level with respect to the complete market depth of back and lay amounts, total matched volume at every different prices, and last matched price.

The set of data for the train and test processes is constructed from the present to the past until the necessary data to train the DL NN models is reached, as observed in Fig. 2. In addition, it also represents the time window defined to re-process data and maintain the DL NN models up to date. In our framework, DL NN models are updated with the new information added every month.

<sup>2</sup> Level II market data provides the additional information needed to trade based on changes that occur in the bid and ask sides. Traders observe the amounts that are being bid against the amounts that are being ask at different prices since this indicates which side is more eager or more powerful, thereby allowing to predict the short-term direction of the price. It is important to emphasize that the historical data that Betfair provides (in files) do not record market depth. For this reason, there is the need to develop a customized capture and store process through the Betfair API in real time as performed in this study.

## 2.2. Characteristics of trading at Betfair

Bettors can bet on (i.e. back) or bet against (i.e. lay) a given horse, and can also submit both market and limit orders. In the context of this study, a market should be interpreted as a platform in which people and entities trade fungible items of value at prices that reflect supply and demand such that the conceptual model consists of a representation of multiple interactions among several participants. Prices are quoted in the form of odds. The pricing ladder ranges from 1.01 to 1000. At this stage, it is important to understand how trade works. The goal of an agent is to secure profit by completing a trade. To complete a trade, two opposing bets need to be placed. Once they are both matched, the trade is closed and a profit or a loss is locked. One of the bets corresponds to a winning bet, while the other corresponds to a losing bet. The profit of a back bet is calculated using Eq. (1) and the liability (i.e. amount in case of a loss) of a back bet is the amount of the bet itself.

$$\text{Profit Back Bet} = \text{Amount} \times (\text{Price} - 1) \tag{1}$$

The liability of a lay bet is given by Eq. (2) and the profit is the amount of the bet itself. Basically, the lay bet is the mirror of the back bet.

$$\text{Liability Lay Bet} = \text{Amount} \times (\text{Price} - 1) \tag{2}$$

PL is given by Eq. (3) and it corresponds to the profit obtained with the winning bet minus the loss obtained with the losing bet:

$$PL = \text{Profit of winning bet} - \text{Liability of losing bet} \tag{3}$$

Consider the example of a trade where it is unnecessary to know the end result of an event to secure a given PL: back of £2@2.12 and lay of £2@2.10. For a bet to be matched, it must become the best offer in the market and it has to be purchased with a counter bet. When the runner is a winner, the profit (back bet) – loss (lay

bet) is:

$$2 \times (2.12 - 1) - 2 \times (2.10 - 1) = 2.24 - 2.20 = \text{£}0.04$$

When the runner is a loser, the profit (lay bet) – loss (back bet) is:

$$2 - 2 = \text{£}0$$

Note that if we have this kind of back/lay bet combination with the same amount at different prices there will be profit (when the back price is higher than the lay price) or loss (when the back price is lower than the lay price) as long as the runner in question wins the event. Otherwise, if any other runner wins the event and the combination of back/lay bets have the same amount, then the PL will be null. To ensure a given PL amount whatever the final outcome, this has to be distributed across all runners, a process designated by *greening* or *hedging*. The amount to close the trade must be then recalculated. If a back order is open on the market, the amount to close the position with the corresponding lay order is calculated using Eq. (4):

$$\text{Close Amount Lay} = \frac{\text{Price Open in Back}}{\text{Price Lay to Close}} \times \text{Amount Open in Back} \quad (4)$$

If a lay order is open on the market, the amount to close the position with the corresponded back is calculated using Eq. (5).

$$\text{Close Amount Back} = \frac{\text{Price Open in Lay}}{\text{Price Back to Close}} \times \text{Amount Open in Lay} \quad (5)$$

Finally, it is relevant to emphasize that Betfair does not charge transaction fees but rather profit fees (i.e. fees applied to the total profit obtained by the bettor) at the end of an event. Therefore, in the context of this study, the spread or transaction cost per bet executed is null, which clearly favors high frequency trading. As soon as a given bettor reaches some consistent level of earnings, he/she becomes also subject to premium charges over profits with a weekly incidence.

### 3. Applied deep learning architectures

DL is based on NNs. A NN is a class of algorithms composed by *MP-Units* (i.e. neurons) firstly introduced by McCulloch and Pitts (1943), as clarified in Fig. 3.

The computation of a single neuron C is given by:

$$C(X, \theta) = \phi \left( \sum_{n=1}^N (w_n \cdot x_n) \right) \quad (6)$$

The parameter  $\theta$  represents the set of weights  $\{w_1, \dots, w_n\}$  and the vector  $X$  represents the set of inputs. The function  $\phi$  represents the activation function,  $w_n$  the weight on connection  $n$ , and  $x_n$  the input value  $n$  in the neuron. Traditionally, activation functions were taken to be smoothed-out step functions (e.g., sigmoid). A NN is composed by several layers with multiple neurons. The goal of a NN is to create a mapping function  $\hat{f}$  that approximates the final

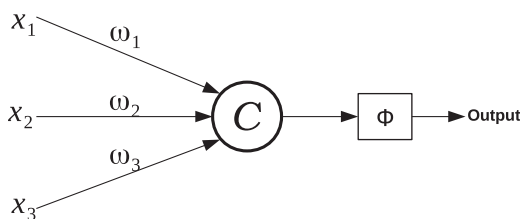


Fig. 3. Single perceptron, unit or neuron architecture.

output to the real target  $f$  by minimizing an error function  $J$ . In DL, the backbone process to find the ideal  $\theta$  is called backpropagation (Rumelhart et al., 1985). The goal of backpropagation is to iteratively compute the gradient descent, that is, the partial derivatives of the error function  $J(\theta_i)$  with respect to all weights  $w$  (being the set of all  $w$  on iteration  $i$ :  $\theta_i$ ) and propagates the error from the output layer to the input layer correcting the weights with an optimizer rule. The efficient adaptive moment estimation optimizer (Adam) is adopted since it is currently considered the best practice (Kingma and Ba, 2014). At the output layer, this study deals with a time series classification problem. This means that we have a number of output classes  $N$  and the model will assign a probability for each class  $n$  (i.e. strong up, weak up, neutral, weak down, and strong down). The error function  $J$  considered in this study is the categorical cross entropy (i.e. logarithmic loss):

$$J(f, \hat{f}) = - \sum_{n=1}^N f_n \ln(\hat{f}_n) \quad (7)$$

It allows to optimize the learning process penalizing wrong guesses by encapsulating the difference between the network output probabilities and the true class  $n$ , with  $n = \{1, \dots, N\}$  and  $N$  the total number of classes to predict.

The term deep in NNs emerges from the deepness in the number of intermediate layers between the input layer and the output layer. Thus, a NN is deep if there are several hidden layers. Deepness may cause the vanishing gradient problem whenever using backpropagation in the training process. This is because, at each learning iteration, each weight receives an update proportional to the gradient of the error function. The vanishing gradient problem occurs when the error signal that passes backwards starts approaching zero. Formally, this occurs when the derivative  $\phi'$  of the activation function  $\phi$  is close to zero, especially for saturated neurons.<sup>3</sup> By iteratively throwing the error signal backwards, it becomes weaker and, hence, vanishes. Contrary to classical activation functions (e.g., ArcTan, sigmoid), the relatively new Rectified Linear Unit (ReLU) activation function:

$$\phi = \max(0, x) \quad (8)$$

solves this concern (Arora et al., 2016). ReLU not only alleviates saturation issues by mitigating the vanishing gradient problem, but also forms highly sparse DL NNs, given that the derivative  $\phi'$  takes constant value 0 for  $x < 0$ , while taking value 1 for any positive value of the input, thereby inducing a more efficient training and reliable test results. Since we work with a time series classification problem, the output layer with  $N$  neurons uses the softmax activation function:

$$\phi_{\text{out}}(x_m) = \frac{\exp(x_m)}{\sum_{n=1}^N \exp(x_n)} \quad (9)$$

where  $m$  is a specific output neuron of a class. This activation function ensures that such a purpose is met since it allows to represent a probability distribution over the different possible predicted classes such that the sum of all probabilities is equal to 1 (LeCun et al., 2015).

A systematic problem in NNs with multiple layers and many neurons is overfitting since the network overly memorizes details of the training set such that it ends up performing poorly on

<sup>3</sup> A saturated neuron is a neuron in which nodes have outputs close to the extreme values of an activation function. Saturation leads to a situation where a change in one input-to-hidden weights during training is unlikely to change the sum-of-products such that, after activation, the node output will remain at the extreme values (LeCun et al., 2012).

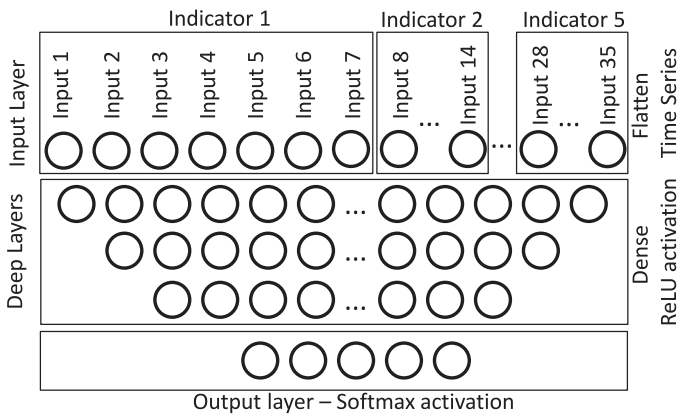


Fig. 4. Illustration of DNN architecture.

the test set. Regularization refers to methods used to avoid overfitting. An important method is the dropout, which was applied to intermediate layers since it allows to ignore a random subset of neurons during each training update in order to avoid overdependence (see Table 3 in Section 4.3.1). Let us explain in detail the three DL NN architectures considered in this study.

### 3.1. Deep Neural Network Classifier

A DNNC is a type of Feed-Forward Neural Network (FFNN) that contains intermediate fully connected layers. In a FFNN, the layers are ordered and each unit of a layer connects to the units of previous layers. As clarified in Fig. 4, the DNNC employed in this study is constituted by an input layer, three intermediate layers using ReLU activation function, and an output layer using softmax activation function. The DNNC should be interpreted as a mere benchmark relative to the alternative DL NN architectures.

### 3.2. Long Short-Term Memory

RNNs and, in particular, LSTMs are state-of-the-art approaches for time series forecasting. Their efficiency can be explained by recurrent connections that allow the network to access the history of previous time series values. This means that a RNN is characterized by connections that have loops, adding feedback and memory to the network over sequences which allows to learn and generalize across sequences of inputs rather than through individual patterns. LSTM is capable of learning order dependence in sequence prediction problems and solving seamlessly problem settings composed by multiple time series input variables (Bengio et al., 1994).

LSTMs contain cycles that recursively feed the network activations from a previous time step as inputs to the network in order to influence predictions at the current time step. This mechanism allows to exploit a dynamically changing environmental window over the input sequence history (Sak et al., 2014). Moreover, LSTMs add a cell state to the basic RNN that runs straight down the entire recursive chain, with only some minor linear interactions in order to control the information that needs to be remembered. LSTMs also have the ability to remove or add information to the cell state by means of structures called gates. The hidden state of a LSTM is composed by memory cell, input gate, output gate, and forget gate. The memory cell stores a value or state, for either long or short time periods. This is achieved by using an activation function for the memory cell (e.g., sigmoid). The input gate controls the extent to which a new value flows into the cell. The forget gate controls the extent to which a value remains in the cell. The output gate controls the extent to which the value in the cell is used to compute the output activation of the unit. Gates have in and out con-

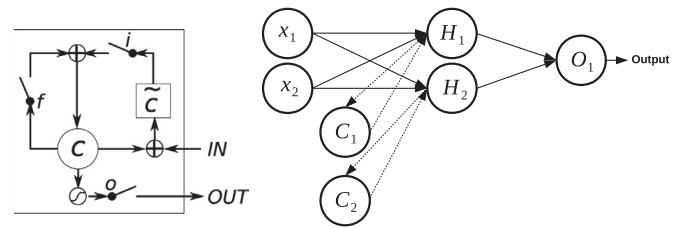


Fig. 5. LSTM hidden state mechanism on the left:  $i$ ,  $f$  and  $o$  correspond to the input, forget and output gates, respectively.  $c$  and  $\tilde{c}$  denote memory cell and memory cell update value, respectively. Illustration of LSTM recursive loop on the right.

nections. The respective weights, which need to be learned during training, are used to orient the operation of the gates. The underlying mechanism is illustrated in Fig. 5.

The LSTM network employed in this study is constituted by two intermediate layers, a LSTM and a dense. For classification, we consider the output layer with a softmax activation function (see Table 3 in Section 4.3.1).

### 3.3. Convolutional Neural Network

CNNs have gained popularity due to their success in classification problems. CNNs, which are traditionally applied to image classification, use an ad hoc architecture inspired by biological data taken from physiological experiments done on the visual cortex (Zeiler and Fergus, 2014).

Convolutional layers are comprised of filters (i.e. kernels or weight matrices), which are the neurons of the layer, that have weights and provide an output. The input and output of each layer are designated by feature maps. Hence, a feature map is the output of a filter applied to the previous layer. An output is obtained by sliding a filter over the input and at each point computing the product between the receptive field area of the input and the filter. The distance of 'pixels' traveled by the filter is called stride. Each position of the filter results in the activation of the neuron such that this structure allows the model to learn filters that are able to recognize specific patterns in input data. Pooling layers down-sample the previously generated feature maps. They follow a sequence of one or more convolutional layers in order to consolidate the features learned and expressed in the generated feature map (LeCun et al., 2012). As such, they may be considered a technique to compress or generalize feature representations and generally reduce overfitting in the training set. By normally taking the average or maximum of the input area, pooling layers mechanically create feature maps (i.e. not being subject to learning).

After passing through a sequence of convolutional and pooling layers, the resulting feature maps are flattened (i.e. the multidimensional information is flattened into one dimension) and fully connected layers are used, just as in a DNNC, to finalize the classification process. These fully connected layers are then used to create final non-linear combinations of features, thereby allowing to make classifications. CNNs are suitable to our case study given that multivariate time series analysis can be seen as a bi-dimensional input likewise an image classification problem. In image recognition, CNNs preserve the spatial relationship between pixels to learn internal feature representations. Instead of using correlations between pixels, CNNs in the context of time series learn patterns considering correlations between indicators and time segments (see the input of the CNN in Fig. 6). Hence, the application of CNNs to time series forecasting means to learn filters that represent certain patterns in the time series and use these to forecast future values. Despite the imminent advantage in their use, the

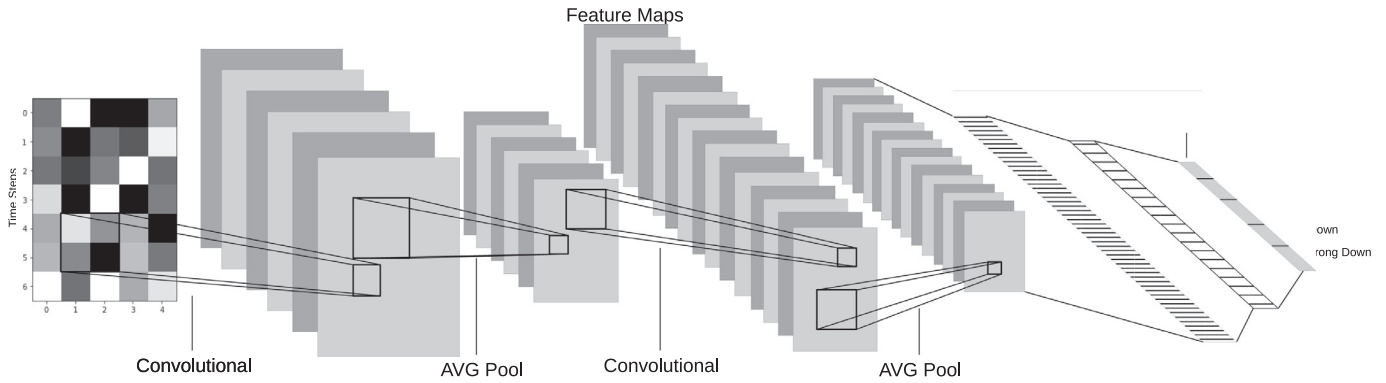


Fig. 6. Illustration of CNN architecture.

application of CNNs in exchange markets is still scarce. Fig. 6 illustrates the three types of layers (i.e. building blocks) usually observed in CNNs: convolutional layers, pooling layers and fully connected layers.

We consider two convolutional layers, one average polling layer, two dense layers and an output layer. In the first convolutional layer, a  $2 \times 5$  filter size is considered for a  $5 \times 7$  bi-dimensional input size. The second convolutional layer considers a  $3 \times 3$  filter size. The pooling layer is used as a simple temporal down-sampling where each unit takes the average value in a  $1 \times 2$  area size. The parameterization considered is systematized in Table 3.

4. Methodology

In order to classify price movements, we must identify the type of market dynamics, define the input indicators and output classes, and parameterize the DL NN models.

4.1. Categories: data partition

The goal of implementing a rule-based decision tree is to categorize the different market dynamics faced by the system. Table 2

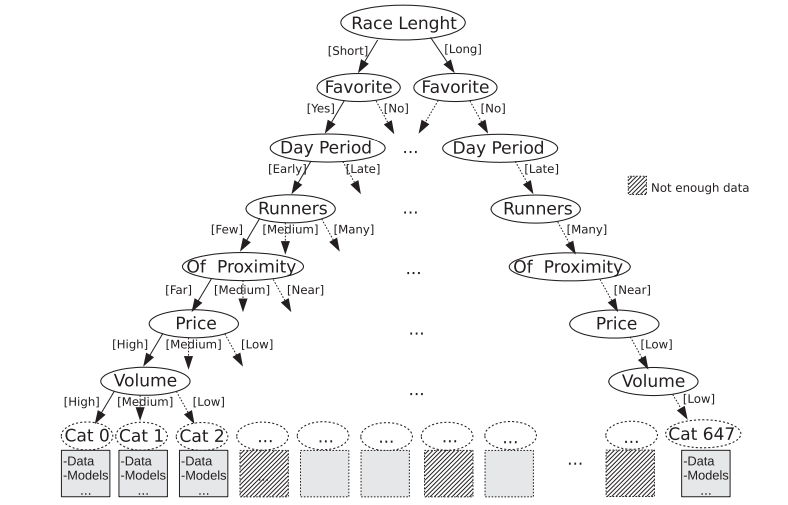
systematizes the decision tree implemented in this study, which was advised by experts. Accordingly, the market under analysis is categorized based on:

1. Races with short or long length;
2. Races with clear favorites;
3. Early or last races of the day;
4. Races with low, intermediate or high number of runners;
5. Pre-live moment: far or near the start of a race;
6. Trading runner has low, medium or high odds;
7. Trading runner has low, medium or high volume transacted.

With regard to the first rule, traders face a higher (lower) risk in short (long) races if deciding to close a position after the beginning of a race, respectively. The second rule is justified by the fact that the price variation on a clear favorite has implications on the variation of price in all other runners. Models within this category (i.e. with this property) incorporate the price evolution of the clear favorite in their input indicators. The third rule is justified by a higher number of recreational bets in the last races of the day since these occur after the end of daily work journeys, which affects the market dynamics. The fourth rule is justified by

Table 2 Rule-based decision tree.

Race length (1)	Favorite (2)	Day period (3)	Runners (4)	Time proximity (5)	Price (6)	Volume (7)
Short	Yes	Early	Few	Far	High	High
Long	No	Late	Medium	Medium	Medium	Medium
			Many	Near	Low	Low



**Table 3**  
Parameterization of each DL NN architecture.

DNNC			LSTM			CNN		
Input (35)			Input (7,5)			Input (7,5)		
Layer	Size	Dropout	Layer	Size	Dropout	Layer	Size	Dropout
Dense	100	0.4	LSTM	100	0.4	Conv2D	20 [2,5]	null
Dense	50	0.2	Dense	100	0.3	AVGPool	[1,2]	null
Dense	20	0.1	Output Dense	5	null	Conv2D	40 [3,3]	null
Output Dense	5	null				Dense	400	0.5
						Dense	140	0.8
						Output Dense	5	null

(Past) Frame 3				Frame 2				Frame 1 (Present)			
Bid	Price	Ask	Volume	Bid	Price	Ask	Volume	Bid	Price	Ask	Volume
	!				!				!		
	4,8	263	24		4,8	263	24		4,8	263	24
	4,7	148	70		4,7	148	70		4,7	148	70
	4,6	349	76		4,6	349	76		4,6	349	76
8	4,5				4,5	92	8		4,5	92	8
2	4,4			2	4,4				4,4	98	2
10	4,3			10	4,3				4,3		
448	4,2			448	4,2			448	4,2		
	!				!				!		

**Fig. 7.** Market evolution in a 3 RDFs time segment.

the distribution of the total volume transacted across the number of runners, which also affects the market dynamics. In addition, in races with many runners the price movement of a particular runner has low impact on others compared to races with a low number of runners. In our case study, few defines races with 5 or fewer horses, medium defines races with 6 to 10 horses, and high defines races with 11 or more horses. The fifth rule has already been explained and highlighted in Fig. 1. Both average trading volume and average liquidity increase from the 10th minute before the race starts until the minute before the start of a race, which clearly affects the market dynamics. The sixth rule defines categories based on whether odds are low, medium or high in the runner under analysis. Different range odds reflect different dynamics. In this study, low corresponds to odds between 1.01 and 3.50, medium corresponds to odds between 3.55 and 6.00, and high corresponds to odds between 6.20 and 12.00. We neglected runners whose odds are above 12.00. Finally, the seventh rule is concerned with the volume matched in the runner under analysis.

The combination of the properties listed in Table 2 generates a total of 648 different categories (i.e. tree leaves), which are indexed to simplify data treatment. However, only 240 categories satisfy the minimum amount of data required to train the DL NN models. These correspond to the most frequent market states.

## 4.2. Feature engineering

### 4.2.1. Definition of inputs and outputs

Five indicators are selected as inputs of the DL NN models, namely:

1. Integral of the price change of the runner in trade;
2. Integral of the price change of the competitor runner;
3. Liquidity variation in the ask side;
4. Liquidity variation in the bid side;
5. Volume variation and direction;

The first indicator is measured by the sum of the number of ticks variation. The second indicator is given by similar unit measure, although with the proviso that the competitor horse is assumed to be the one whose odd is closest to the price of the runner in trade. However, if there is a clear favorite, then it is con-

sidered the competitor of runner in trade since this corresponds to the runner with most influence on its price. The third (fourth) indicator is measured by the variation of lay (back) amounts waiting to be matched, respectively.

Finally, the fifth indicator is the volume variation and direction, which is representative of the amount of money that has been matched during a time segment. If limit orders on the ask side are absorbed, then the volume variation is positive, while being negative otherwise. In order to provide a higher clarity on how each indicator evolves over time, consider the following discrete time market dynamics for a 3 RDFs time segment<sup>4</sup> exemplified in Fig. 7.

One can observe that the last matched price is 4.6 in frame 3 (i.e. at time  $t - 2$ ). Suppose that runner  $x$  is gaining favoritism such that there is a tendency in the market to pull the odd down. As long as back market orders enter the market, the price falls. The last matched price becomes 4.5 in frame 2 (i.e. at time  $t - 1$ ) and then 4.4 in frame 1 (i.e. at time  $t$ ). As shown in Fig. 8, this market dynamics has a direct impact on each selected indicator.

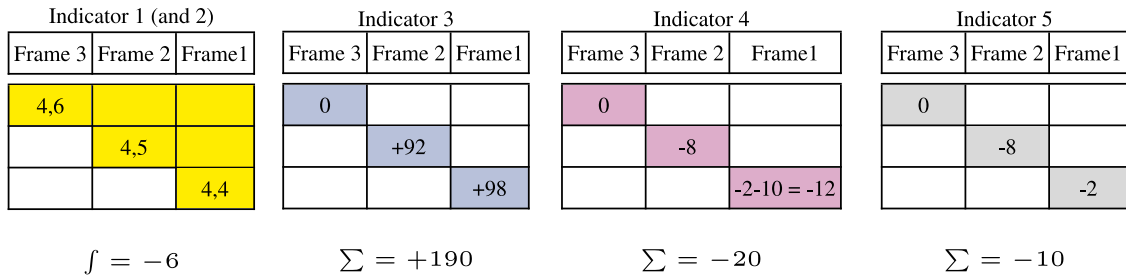
From time  $t - 2$  to time  $t$ , we observe that:

1. The odd ranged 2 ticks down in 2 time steps, hence, the integral of ticks is equal to  $-6$  in the runner in trade;
2. Mutatis mutandis, the graphical representation of the ladder associated with the competitor runner is omitted given that the intuition is qualitatively similar to that of indicator 1;
3. The variation of back amounts is equal to  $+190$ ;
4. The variation of lay amounts is equal to  $-20$  due to the inclusion of the back amount of 10 canceled out at 4.3;
5. The volume variation is given by the  $\text{€}8$  from time  $t - 2$  to time  $t - 1$  plus the  $\text{€}2$  from time  $t - 1$  to time  $t$ . The volume direction is negative because the last matched price decreased.

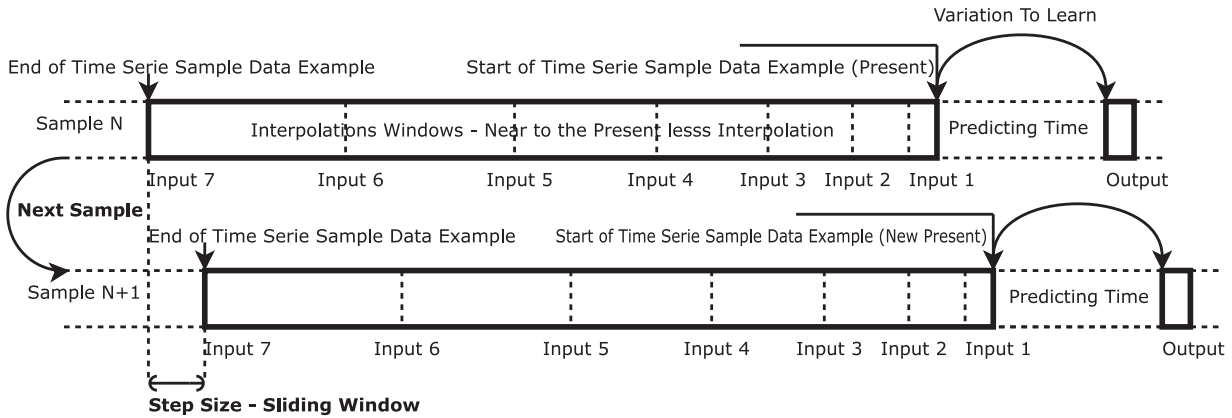
This simple illustration allows to observe how the 3 RDFs time segment is compressed into a single value for each indicator.

In our case study, these 5 indicators were used, each one composed by 7 compressed time segments and, thus, the total number of inputs used to train and test the DL NN models is equal to

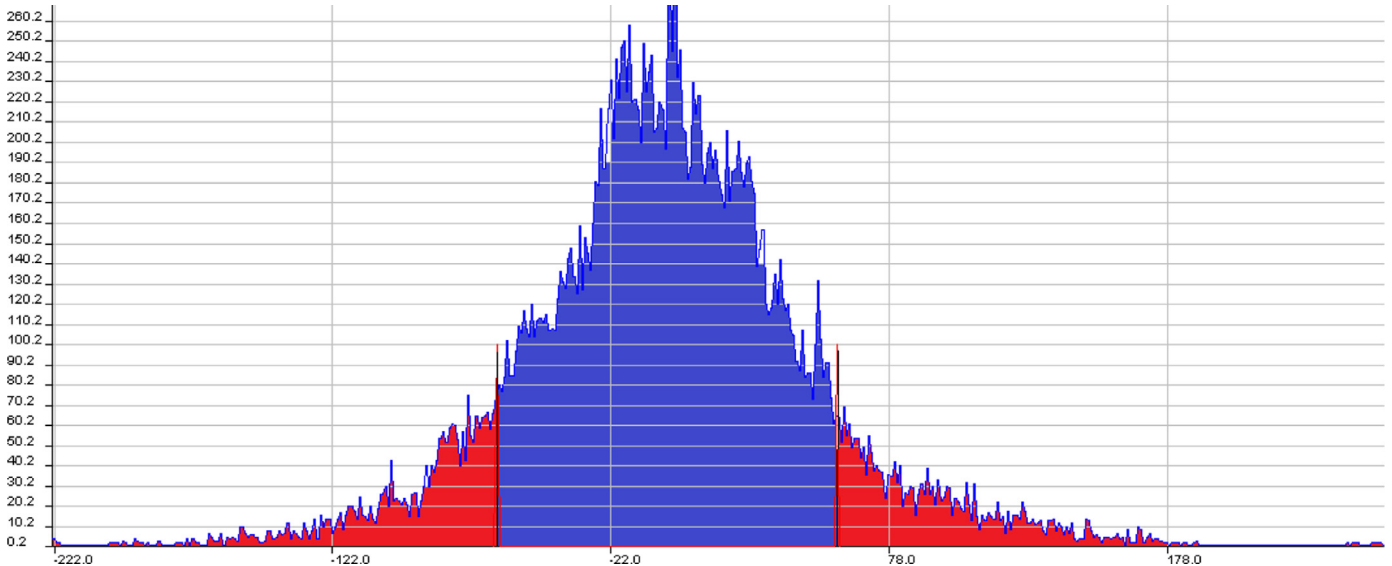
<sup>4</sup> A time segment is a sequence constituted by a number of RDFs.



**Fig. 8.** Impact of the market dynamics on each selected indicator. Note: At frame 1 in Fig. 7 the lay amount of 10 at 4.3 disappears not due to the matching process, but because it was canceled out given that the volume information did not change.



**Fig. 9.** Time series sampling for a given indicator.



**Fig. 10.** Example of one input variable histogram re-scaling with truncated tails at 10% level to find min–max values for input normalization.

35 (i.e.  $5 \times 7$ ). Moreover, the size of the time segments follows a non-linear process as clarified in Fig. 9 and Table A.2. Near present time segments have smaller sizes compared to past time segments in order to provide a higher precision of information to the DL NN models.<sup>5</sup>

The output or target corresponds to the price variation between time  $t$  and time  $t + n$  RDFs ahead, with  $n = 90$ . As explained in Subsection 4.2.2, this variation is converted into classes meaning that the case study deals with a classification problem.

<sup>5</sup> In our specific problem, a non-linear time segmentation process means that more relevance is given to the nearest past through the application of an exponential factor (see Table A.2 in the Appendix).

#### 4.2.2. Frequency distribution histograms

The input values of the DL NN models are normalized and outliers can substantially affect the ability of DL NN models to learn properly. One of the characteristics of learning algorithms is that they tend to smooth out noise. This is good in the sense that it allows an effective modeling of noisy systems. However, Deboeck (1994) claims that, if most data are concentrated in a very small portion of the input range, then inputs may have little effect on the resulting model. To solve this problem, outliers are truncated based on the application of frequency analysis and histogram re-scaling. Fig. 10 illustrates an example of the automatic process of truncation of outliers. This operation is systematically applied to all inputs for each category.



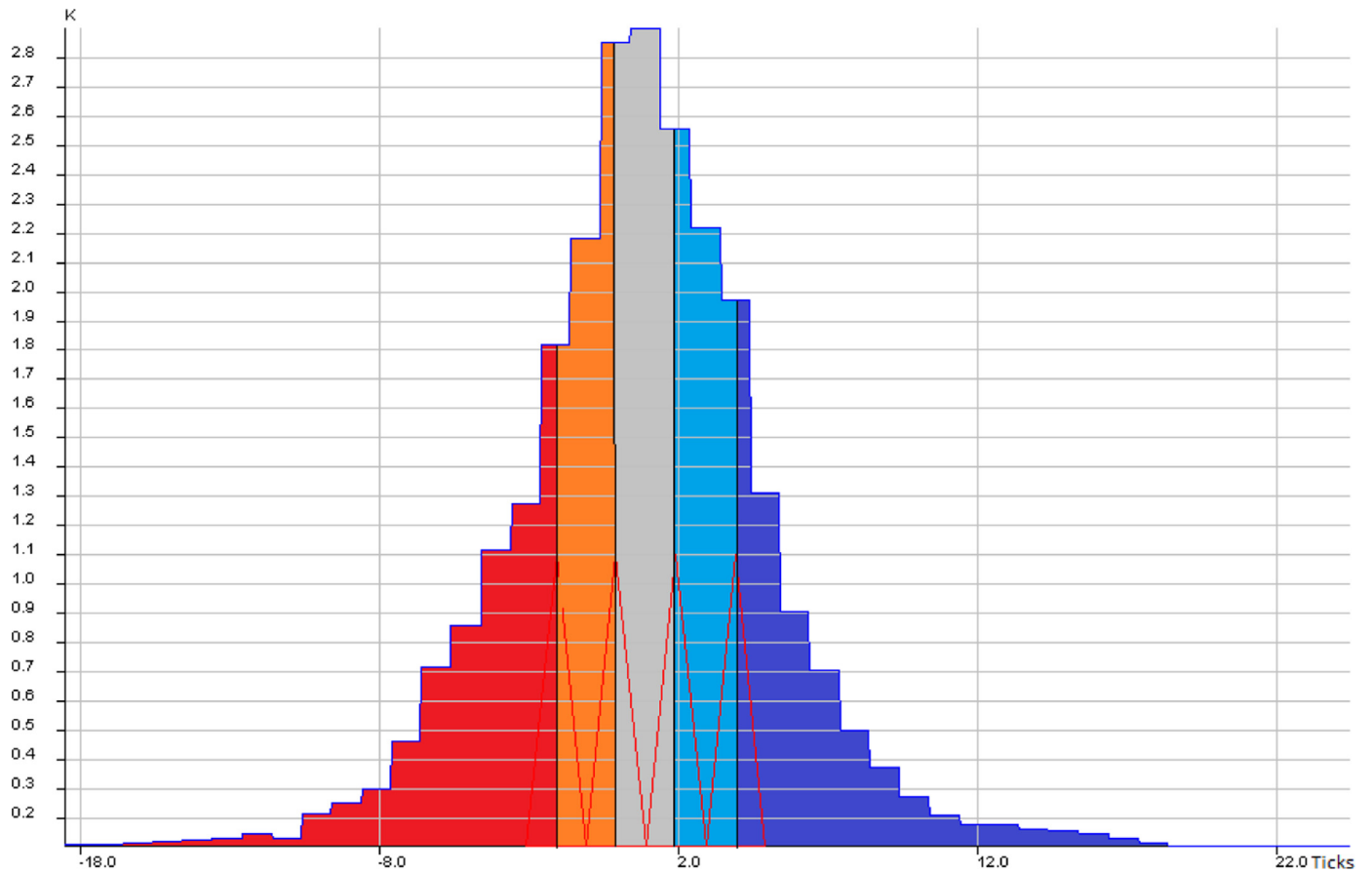


Fig. 11. Histogram for the qualitative classification of the output: strong down (red), weak down (orange), neutral (grey), weak up (light blue) and strong up (dark blue), respectively. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

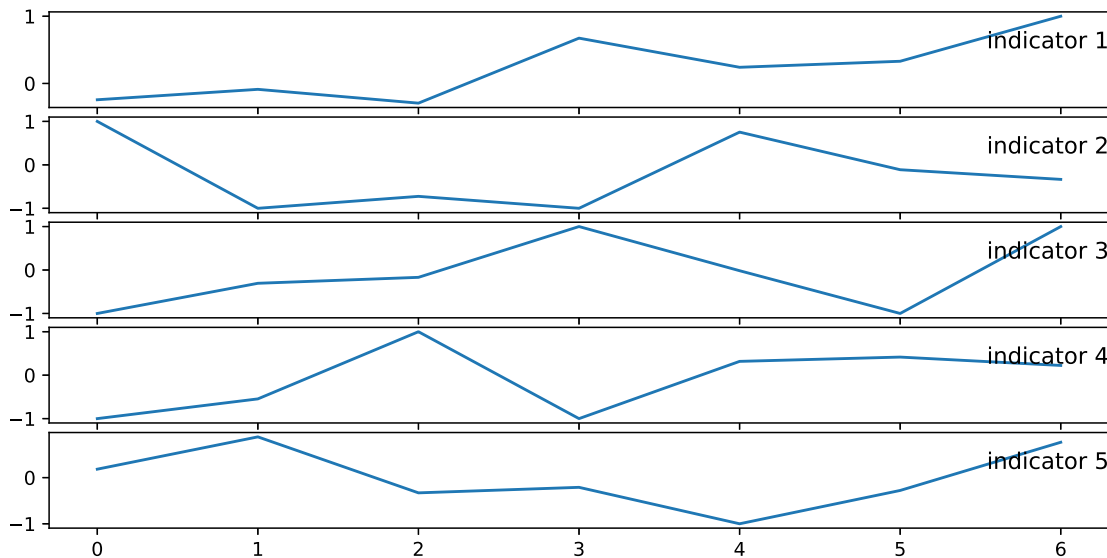


Fig. 12. Sample example of 5 indicators with 7 time segments that corresponds to a strong up class.

As clarified in Fig. 11, similar technique is used at the output level to transform the regression problem into a classification problem. An egalitarian distribution in the number of examples per qualitative class is defined. The output categories considered in this study are given as follows. When the numerical solution falls into the first qualitative class, a strong down price change is established. When the numerical solution falls into the second qualitative class, a weak down price change is considered. When the numerical solution falls into the third qualitative

class, there is a neutral price change. When the numerical solution falls into the fourth qualitative class, we have a weak up price change. Finally, a strong up price change occurs when the numerical solution falls into the fifth qualitative class.

In summary, raw data are transformed to ensure that DL NN models receive 5 normalized time series indicators with 7 compressed time segments in order to perform classification on this input as clarified in Fig. 12.

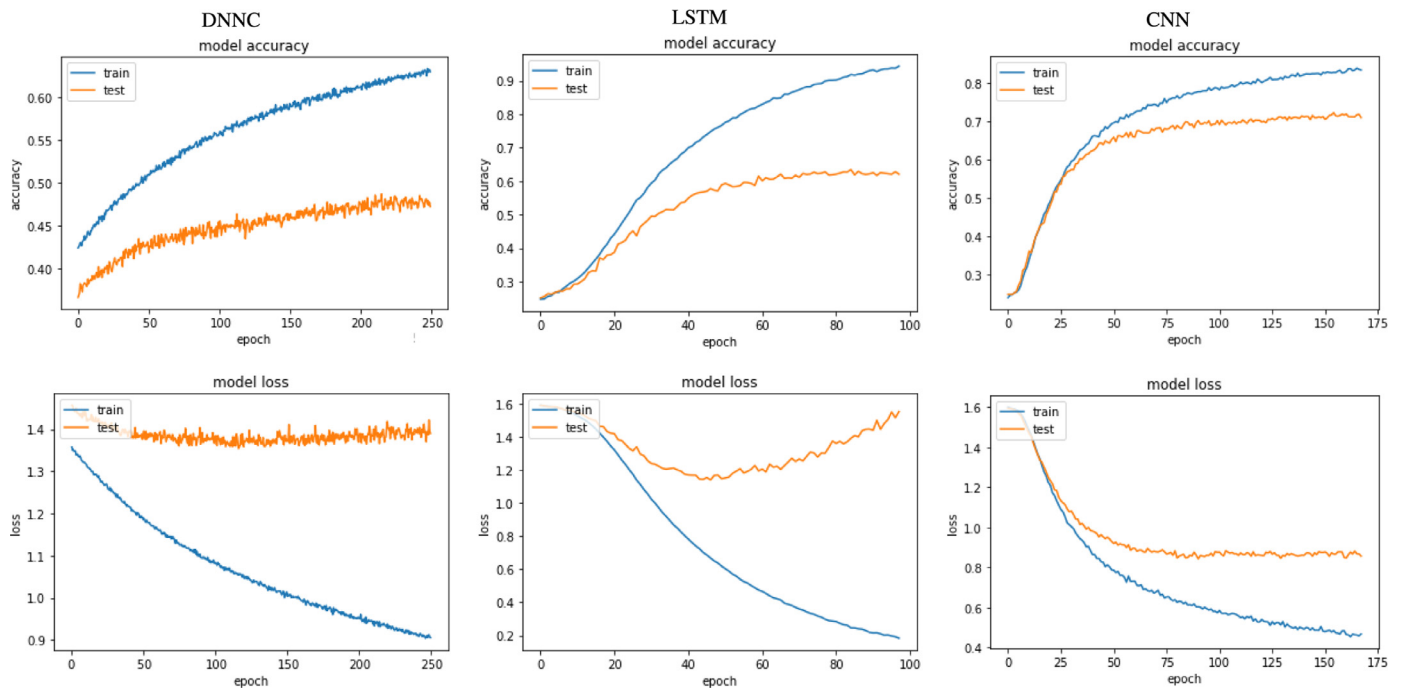


Fig. 13. Learning and test results in a particular category.

### 4.3. Modeling

The transformed dataset, ready to feed the DL NN models, is split into two subsets for the purpose of learning: training and testing. In this study, 80–20% partition of data between train and test sets is considered for each category. The training set is used to adjust the weights of the DL NN models to allow the recognition of patterns. The test set confirms the actual predictive power of each trained DL NN model, which means that models are tested through a certain measure of performance (e.g., accuracy) to understand whether they can generalize to new data.

#### 4.3.1. Training process

The training process consists of the periodic treatment of all DL NN models for each of the 240 available categories. The maximum training degree is defined by the number of iterations, which is set at 400 epochs. However, we define an Early Stopping (ES) mechanism, which essentially aims at maximizing the level of accuracy in the test set. If the accuracy does not improve (i.e. either remains constant or decreases) in the last 12 epochs, then the ES mechanism automatically stops the training. This test is applied in every training epoch. Common properties are applied to all DL NN models in every categories in the training phase (i.e. crossentropy error function is considered and softmax activation function is applied to the output layer). Finally, we set a batch size of 64. For each DL NN architecture, a grid search is applied to find the optimal value of some hyperparameters, namely layers used, size (i.e. number of neurons in the dense layers under DNNC, number of units under LSTM, filters and respective sizes under CNN), and dropout level in the different layers. Table 3 compiles the parameterization considered for each DL NN architecture.

A final caveat is worth explaining. With a DNNC, inputs are flattened, introduced into the first fully connected dense layer, and their shape is one-dimensional with 35 features. With the LSTM and CNN as well, inputs have a bi-dimensional shape characterized by 7 time segments  $\times$  5 indicators. Finally, for each category, the historical training process is stored.

#### 4.3.2. Testing process

Performance evaluation of the trained DL NN models is developed through the observation of accuracy and loss metrics.

Fig. 13 shows an example of the test results in one of the 240 categories. Accuracy and loss are displayed for each DL NN model. Although both are usually presented at the train set level (represented by blue color) and at the test set level (represented by orange color), more attention should be given to the latter representation. At this point, it is important to know how to interpret the above subplots. Three main conclusions should be emphasized with respect to the test results. First, the DNNC has the lowest level of accuracy. In addition, it corresponds to the DL NN architecture that takes more time to learn based on the number of processed epochs, thereby confirming that its adoption is ill-advised in relation to the remaining alternatives. Second, the LSTM is the first DL NN model reaching the ES rule. This is visible by the fact that it has the lowest number of processed epochs. As previously mentioned, the ES rule maximizes the level of accuracy in the test set. If, instead, loss minimization was considered, the LSTM would reach the ES rule earlier, thereby suggesting that this DL NN architecture may suffer from some degree of overfitting. Third, the CNN provides the highest level of accuracy, even though it requires a superior number of epochs to reach the steady-state compared to the LSTM. This result is consistent regardless whether accuracy is maximized or loss is minimized.

### 4.4. Validation with models in production

Depending on the class of price movement predicted by the DL NN models, different trading mechanisms can be activated. In addition, the parameterization (e.g. stop-loss and target) of the trading mechanisms depends on the category in which the runner belongs to.

#### 4.4.1. Definition of the trading mechanisms

The betting strategy consists of trading simultaneously in various horses for which there is a valid model (see Table 2). Based

**Table 4**  
Example of trading mechanism parameters for a particular category.

Class	Mean of the ticks variation	Target	Stop-loss
Strong Up	6.44794	6	4
Weak Up	3.51428	4	3
Weak Down	-3.19424	3	2
Strong Down	-6.33173	6	4

on the predictions of the DL NN models, a trading mechanism is activated by horse.

The trading mechanisms considered in this study are the swing and trailing stop. Gonçalves et al. (2013) provides the formal treatment and the Java implementation of both trading mechanisms. We take as a given that, when the DL NN models predict either a weak (strong) up or a weak (strong) down price movement, the swing (trailing stop) trading mechanism is activated, respectively.

Swing is similar to scalping the market tick by tick, but with the difference that it can involve more than one tick variation. A back or lay bet is placed at the current odd and a counter bet is placed several ticks offset. The stop-loss is normally defined as a percentage of this offset. Consequently, the swing trading mechanism consists of an algorithm implemented to close a position at the target odd  $x + d$  given some bet placed at the current odd  $x$ , where  $d$  represents the gap between the current and target odds measured in ticks, with  $d \in \mathbb{Z}$ . If the odd varies in the opposite direction to the target odd, a stop-loss is activated at the odd  $x - s$ , where  $s$  represents the gap between the current and stop-loss odds measured in ticks, with  $s \in \mathbb{Z}$ . By using a stop-loss, an agent fixes the value based on the maximum loss he/she is willing to incur. If the odd drops below this value, the stop-loss turns into a market order and will be triggered. Once the current odd falls below the stop-loss odd, the position is closed which prevents any further losses.

The difference between the swing trading mechanism and the trailing stop mechanism is at the level of the stop-loss behavior. While a stop-loss under the swing trading mechanism has a fixed value, a stop-loss under the trailing stop trading mechanism automatically shadows the price movement in the predicted direction, thus, following the price action only when it moves in the predicted direction. Consequently, the stop-loss under the trailing stop trading mechanism self-adjusts. Eventually, the odd will move in the reverse direction and reaches the updated stop-loss odd. In our implementation the trailing stop is also parameterized with a target odd offset, which is larger than the one used in swing. This way if a large movement occurs we do not have to wait for the turn of the market to close in the updated stop price.

We also introduced the parameterization of times at the level of trading mechanisms. In short, if both trading mechanisms do not reach the target odd within one minute (defined for this case study), the position is closed at the market odd. Moreover, in case of any trading mechanism being executed near the start of a race, all positions are closed through the *forceclose()* method 5 s before the effective start of the race. The choice of target and stop-loss odds in each category depends on the de-normalization of the output based on the histogram previously presented in Fig. 11.

The main idea is that this process allows to adjust the parameterization of the trading mechanisms (target and stop-loss) according to each category. The target odd corresponds to the average of the maximum tick variation for all examples of the collected data falling at a given class during the predicting time (see Fig. 9). The stop-loss is defined as 80% of the target odd for swings and 60% of the target odd for trailing stop. Table 4 presents an example of

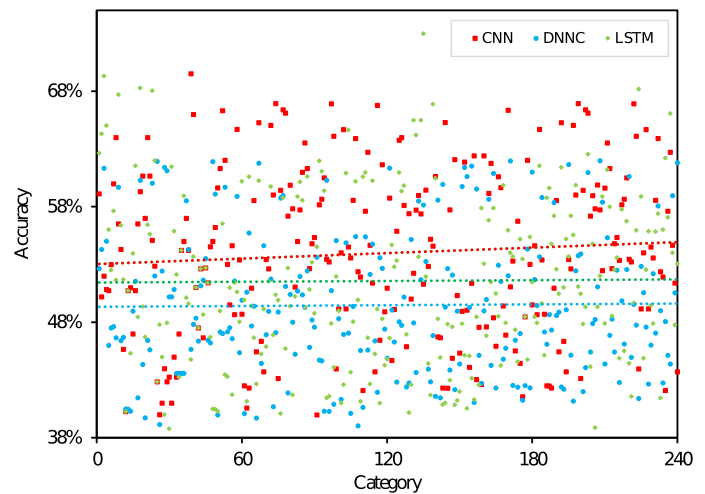


Fig. 14. Test results.

the definition of target and stop-loss odds for each class of a given category.

#### 4.4.2. Experiment

After collecting data, training and testing DL NN models, a simulation of 30 days is conducted. Hence, validation is related to experimentation. The validation set is composed by data collected 30 days in advance relatively to the period of training and testing. The simulation uses trading orders of £3.00. This amount is unable to influence the market trend if used in real interaction with the market. Trading mechanisms are executed after 60 votes retrieved from the models into a FIFO buffer. Each vote is generated at a 2 frames per second (FPS/global RDF update) rate. If winning votes fall into the strong up or strong down qualitative class, a trailing stop is executed. If winning votes fall into the weak up or weak down qualitative class, then a swing is executed.

Finally, modeling is developed in Python 3.5. The DL NN architectures are trained and tested with Keras on top of TensorFlow. The training process is based on distributed computing and executed in the Avalanche FEUP cluster. Each DL NN training process is allocated to 4 cores. Training the total number of categories takes approximately 4 processing days. The DL NN models are validated in a simulator developed in Java (Gonçalves et al., 2013).<sup>6</sup> To provide a better intuition, Table 5 presents the log results of one trading execution.

## 5. Results

Let us clarify the test results. Fig. 14 exposes the accuracy by DL NN model in all valid 240 categories.

We conclude that, on average, the CNN performs better than the alternative DL NN models since it holds the highest mean ( $\hat{A}_{CNN} = 0.54$ ,  $\hat{A}_{LSTM} = 0.52$  and  $\hat{A}_{DNNC} = 0.49$ ). Nevertheless, Fig. 14 shows that there are categories in which the LSTM outperforms the CNN. Indeed, the LSTM has the greatest level of standard deviation in accuracy ( $\sigma_{LSTM} = 0.0729$ ,  $\sigma_{CNN} = 0.0725$  and  $\sigma_{DNNC} = 0.0618$ ). The large range observed in all DL NN models is explained by the disparity of the training dataset size among the various categories.

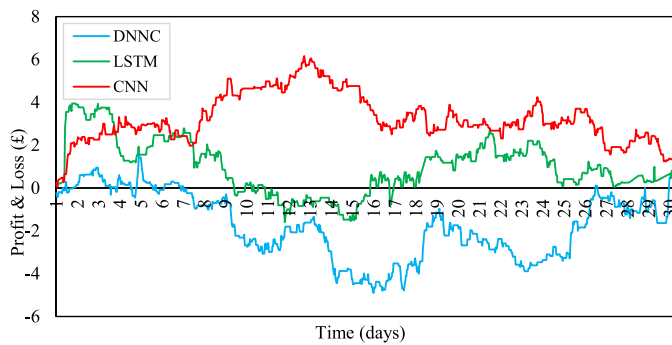
Let us now clarify the validation results. For this purpose, we ignore all categories by each DL NN model whose accuracy is lower than 50%. Fig. 15 exposes the cumulative PL obtained with each DL

<sup>6</sup> See <https://grid.fe.up.pt/> for details on the grid. Source codes are freely available online and can be consulted in this GitHub link: <https://github.com/rjpp>.

**Table 5**

Example of one trading execution log given the category parameters and the model prediction.

Runner category	LongLenght/nofavorite/endingDay/mediumRunners/nearFromBeginning/midleOdd/mediumLiquidity (id:41)
Model predicted probabilities	[0.14, 0.21, 0.17, 0.22, <b>0.26</b> ] (1 vote)
Predicted class	5th class: Strong Up (most voted after 60 votes)
Bets/trade direction	Up: Lay (open) -> Back (close)
Trading mechanism	Trailing Stop (strong movement predicted)
Parameters (in ticks)	Stop-loss: 4, Target: 6
Parameters (in odds)	Entry odd: 4.6, Target odd: 5.2, Stop odd: 4.2
Time parameters	20 frames to open, 80 frames to close
Open amount stake	£3.00 (Lay)
Potential PL	Profit: £0.35, Loss: -£0.28
Trade final state	CLOSED
Moved ticks	6
Close amount stake	£2.65 (Back)
Effective PL	£0.35
Effective close odd	5.2

**Fig. 15.** Validation results.**Table 6**

Global trading simulation results.

	LSTM	CNN	DNNC
Trades	912	1057	1112
Greens	383	455	467
Reaches target	146	145	140
Closes [null; target]	237	310	327
Reds	374	423	461
Reaches stop-loss	236	254	258
Breaks stop-loss	63	80	83
Closes [null; stop-loss]	75	89	120
Null	155	179	184
Positive ticks	1057	1200	1221
Negative ticks	1036	1171	1217

NN model. All reveal a positive PL at the end of the 30th day of simulation. In particular, the DNNC, LSTM and CNN ensure £0.26, £0.51 and £1.35, respectively.

There are four ways to complete a trade, which justify the validation results. First, the trade does not end up being executed because we have an open waiting time of 20 frames and the open bet is not corresponded during this time since we trade at the best price with a limit order (i.e. the position is not open with a market order). Second, after the position is open, everything occurs nicely and the position is closed at the target. Third, the market goes against the prediction and the position is closed at the stop-loss. Fourth, we have a close waiting time of 80 frames to close the position by reaching either the target or the stop-loss. If this time expires, then the position is closed at the current market price which may imply either a profit or a loss. Table 6 summarizes the number of executed trades, greens, reds, positive and negative ticks for each DL NN model. The number of trades is the number of times one trading mechanism is instantiated. This value is not

equal across the DL NN models given that no trading mechanism is instantiated when the predicted class is the neutral. Greens is the number of times the trading mechanism closes in profit. Reds is the number of times the trading mechanism had to close in loss. The sum of greens and reds is not equal to the number of trades because sometimes the trading mechanism can close the trade on the same entry price without making a profit or a loss. This can happen when the trading mechanism reaches the timeout exposure. For these cases the result is null. Also, when the opening bet is not matched during the opening time, the result is null. Positive ticks is the number of total ticks that result in profitable trades. Negative ticks is the total number of ticks that result in loss. These are the main values to get conclusions about how well-succeeded a trading policy is.

The number of greens is higher than the number of reds, however, the total number of ticks in profit (i.e. positive ticks minus negative ticks) is closer to zero. This is because many times prices reach the complete offset ticks for the stop-loss but do not reach the total offset ticks for the target until the waiting time expires, thus, closing at the current market price.<sup>7</sup> Besides that, when the market goes abruptly against the prediction and breaks with the stop-loss, the trading mechanisms try to close the position by following the abrupt price movement, which promotes the persistence of a loss. When the price movement is abrupt and goes in the predicted direction, the limit order is waiting to close and ends up being matched at the target. As such, abrupt price movements can break with the stop-loss, but never generate a profit beyond the target.

A random based policy does not result in a near zero profit, but it results in a permanent loss. Indeed, if we set the agent to execute the trading mechanisms in the opposite predicted direction, losses will be even bigger. This suggest that the DL NN models are struggling to ensure a positive PL in the production phase. This justifies the necessity to improve the resilience of the DL NN models that, despite having a good learning capacity in the modeling phase, only ensure a restrictive PL in the production phase.<sup>8</sup>

Table 7 exposes the Pearson's correlation coefficients. There is a positive and statistically significant correlation between the cumulative PL obtained with the DNNC and the cumulative PL obtained with the LSTM. However, the opposite result is applied to

<sup>7</sup> Note that the stop-loss corresponds to 80% or 60% of the target depending on the instantiated trading mechanism (see Section 4.4.1).

<sup>8</sup> Intuitively, one could say that the capture of strong movement patterns would be more easily detected by the DL NN models. However, we have tried to execute trades only on strong movement predictions (i.e. trailing stops) based on the CNN and found that both strong and weak predictions contribute with a positive, though marginal, profit.

**Table 7**  
Pairwise correlations of the validation results.

	DNNC	LSTM	CNN
DNNC	1		
LSTM	0.53700***	1	
CNN	-0.44904***	-0.55301***	1

Note: Symbol \*\*\* denotes statistical significance at the 0.01 level.

the cumulative PL obtained with the CNN when paired with the cumulative PL of the alternative DL NN models, which reinforces the distinctive character of the CNN.

From a technical point of view, we would expect a higher cumulative PL at the last day of the experiment given the level of accuracy observed in the modeling phase. This suggests the need for a future refinement in the parameters of the trading mechanisms, namely at the stop-loss level since this seems to be hit before reaching the prediction suggested by the DL NN models. This can also be mitigated in the feature engineering stage by considering as output the forecast time integral of the ticks variation rather than simply considering the ticks variation, thus, following similar treatment to the one given to indicators 1 and 2 (see Fig. 8). Ensembling the CNN and LSTM models may be advised since these correspond to distinct DL NN architectures as confirmed in Table 7 and, therefore, their combination may improve test results and ensure a better generalization to new data (Zhou et al., 2002). From an economic point of view, the low profit suggests that a regulatory intervention to ban bots is unlikely. As such, the proliferation of automatic trading systems suggests that the regulation in betting exchange markets is likely to be predominantly focused on the quality of service.

**6. Conclusions**

A short-term forecasting system of price changes applied to exchange markets using market depth data and a systematic procedure to enable a fully automated trading system is implemented in this study. DNNC, CNN and LSTM models are trained and tested to understand which one has the best predictive power. Although the LSTM is more suitable for multivariate time series analysis from a theoretical point of view, results indicate that the CNN holds, on average, the highest level of accuracy and, consequently, generalizes better to new data. Future research avenues include the need to provide an analysis focused on specific niches by grouping properties of the rule-based decision tree (e.g., short versus long race categories). Finally, continuous efforts seem mandatory to analyze domains where the combination of regulation and artificial intelligence may ensure social welfare gains (e.g., determination of the socially optimal fee structure).

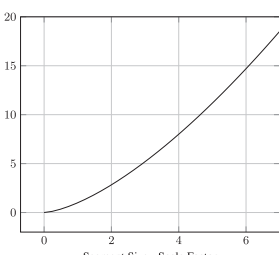
**Appendix**

**Table A.1**  
Summary statistics.

	Mean	Std Dev	Min	Max
<b>Volume</b>				
1st min	520,865	275,541	14,242	4,858,654
2nd min	423,930	247,598	10,783	4,713,870
3rd min	347,772	223,834	8390	4,569,493
4th min	285,066	201,834	7014	4,237,354
5th min	236,840	183,575	5818	3,944,767
6th min	200,076	168,577	4514	3,779,854
7th min	172,516	156,206	2524	3,554,934
8th min	150,905	145,219	1898	3,481,841
9th min	134,880	137,576	1647	3,450,423
10th min	122,693	131,643	1503	3,420,282
<b>Liquidity</b>				
1st min	6974	12,301	531	451,241
2nd min	5967	13,151	682	468,516
3rd min	4975	12,832	526	454,538
4th min	4205	12,544	393	453,012
5th min	3652	12,143	329	428,307
6th min	3105	11,137	285	399,753
7th min	2774	10,808	245	359,948
8th min	2451	9825	212	331,043
9th min	2224	9540	191	323,377
10th min	2029	8935	171	292,921
<b>Volatility<sup>+</sup></b>				
1st min	1052	1443	77	29,899
2nd min	1100	1593	0	31,323
3rd min	1107	1357	22	37,893
4th min	1067	1524	0	29,442
5th min	984	1412	20	33,069
6th min	868	1437	29	49,078
7th min	796	1191	4	30,815
8th min	729	1162	7	27,630
9th min	650	961	2	29,686
10th min	642	1326	1	38,439

Note: Total number of observations (i.e. races): 14,421. Each line represents the x minute before the start of a race, x = {1st,..., 10th}. Units of measure are clarified in Fig. 1. + Ticks variation in absolute value per minute.

**Table A.2**  
Nonlinear process for time segmentation.

$y = x^{factor}$ with $factor = 3/2$	# Segment	Past frame	Time segment size
	1 $\Rightarrow$	0	14
	2 $\Rightarrow$	14	27
	3 $\Rightarrow$	41	34
	4 $\Rightarrow$	75	41
	5 $\Rightarrow$	116	46
	6 $\Rightarrow$	162	52
	7 $\Rightarrow$	214	56

Note: In this example, we consider a total of 270 RDF scaled with exponential factor of 3/2. Segment 1 represents nearest past, while the time segment 7 represents farthest past in the 270 RDF, about 2.5 min, time window.

## References

- Arora, R., Basu, A., Mianjy, P., Mukherjee, A., 2016. Understanding deep neural networks with rectified linear units. arXiv:1611.01491.
- Bengio, Y., Simard, P., Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* 5 (2), 157–166.
- Brown, A., Yang, F., 2017. The role of speculative trade in market efficiency: evidence from a betting exchange. *Rev. Finance* 21 (2), 583–603.
- Deboeck, G., 1994. Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets, 39. John Wiley & Sons.
- Ding, X., Zhang, Y., Liu, T., Duan, J., 2015. Deep learning for event-driven stock prediction. In: *IJCAI*, pp. 2327–2333.
- Dixon, M., Klabjan, D., Bang, J.H., 2015. Implementing deep neural networks for financial market prediction on the intel xeon phi. In: *Proceedings of the 8th Workshop on High Performance Computational Finance*. ACM, p. 6.
- Dorffner, G., 1996. Neural networks for time series processing. *Neural Network World*. Citeseer.
- Fischer, T., Krauss, C., 2018. Deep learning with long short-term memory networks for financial market predictions. *Eur. J. Oper. Res.* 270 (2), 654–669.
- Gonçalves, R., Rocha, A.P., Pereira, F.L., 2013. High level architecture for trading agents in betting exchange markets. In: *Advances in Information Systems and Technologies*. Springer, pp. 497–510.
- Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. *Deep Learning*, 1. MIT Cambridge Press.
- Hatcher, W.G., Yu, W., 2018. A survey of deep learning: platforms, applications and emerging research trends. *IEEE Access* 6, 24411–24432.
- Heaton, J., Polson, N., Witte, J. H., 2016. *Deep learning in finance*. arXiv:1602.06561.
- Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. *Neural Comput.* 9 (8), 1735–1780.
- Hu, Z., Liu, W., Bian, J., Liu, X., Liu, T.-Y., 2018. Listening to chaotic whispers: a deep learning framework for news-oriented stock trend prediction. In: *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, pp. 261–269.
- Huck, N., 2009. Pairs selection and outranking: an application to the s&p 100 index. *Eur. J. Oper. Res.* 196 (2), 819–825.
- Huck, N., 2010. Pairs trading and outranking: the multi-step-ahead forecasting case. *Eur. J. Oper. Res.* 207 (3), 1702–1716.
- Kingma, D.P., Ba, J., 2014. Adam: a method for stochastic optimization. CoRR abs/1412.6980.
- Korczak, J., Hemes, M., 2017. Deep learning for financial time series forecasting in a-trader system. In: *Computer Science and Information Systems (FedCSIS), 2017 Federated Conference on*. IEEE, pp. 905–912.
- LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. *Nature* 521 (7553), 436.
- LeCun, Y.A., Bottou, L., Orr, G.B., Müller, K.-R., 2012. Efficient backprop. In: *Neural Networks: Tricks of the Trade*. Springer, pp. 9–48.
- Mcculloch, W., Pitts, W., 1943. A logical calculus of ideas immanent in nervous activity. *Bull. Math. Biophys.* 5, 127–147.
- Rumelhart, D., Hinton, G., Williams, R., 1985. Learning Internal Representations by Error Propagation. ICS report. Institute for Cognitive Science, University of California, San Diego.
- Sak, H., Senior, A., Beaufays, F., 2014. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv:1402.1128.
- Varian, H.R., 2014. Big data: new tricks for econometrics. *J. Econ. Perspect.* 28 (2), 3–28.
- Zeiler, M.D., Fergus, R., 2014. Visualizing and understanding convolutional networks. In: *European Conference on Computer Vision*. Springer, pp. 818–833.
- Zhao, Y., Li, J., Yu, L., 2017. A deep learning ensemble approach for crude oil price forecasting. *Energy Econ.* 66, 9–16.
- Zhou, Z.-H., Wu, J., Tang, W., 2002. Ensembling neural networks: many could be better than all. *Artif. Intell.* 137 (1–2), 239–263.