Artificial Neural Networks architectures for stock price prediction: comparisons and applications

Luca Di Persio
University of Verona
Department of Computer Science
Strada le Grazie, 15 - Verona
Italy
luca.dipersio@univr.it

Oleksandr Honchar
University of Verona
Department of Computer Science
Strada le Grazie, 15 - Verona
Italy
oleksandr.gonchar@univr.it

Abstract: We present an Artificial Neural Network (ANN) approach to predict stock market indices, particularly with respect to the forecast of their trend movements *up or down*. Exploiting different Neural Networks architectures, we provide numerical analysis of concrete financial time series. In particular, after a brief résumé of the existing literature on the subject, we consider the Multi-layer Perceptron (MLP), the Convolutional Neural Networks (CNN), and the Long Short-Term Memory (LSTM) recurrent neural networks techniques. We focus on the importance of choosing the correct input features, along with their preprocessing, for the specific learning algorithm one wants to use. Eventually, we consider the S&P500 historical time series, predicting trend on the basis of data from the past days, and proposing a novel approach based on combination of wavelets and CNN, which outperforms the basic neural networks ones. We show, that neural networks are able to predict financial time series movements even trained only on plain time series data and propose more ways to improve results.

Key–Words: Artificial neural networks, Multi-layer neural network, Convolutional neural network, Long short-term memory, Recurrent neural network, Deep Learning, Stock markets analysis, Time series analysis, financial forecasting

1 Introduction

During recent years, Artificial Neural Networks (ANN) have been interested by a renewed attention by academicians, as well as by practitioners, particularly within the financial arena. The turning point being represented by the winning solution for *ImageNet challenge*, in 2012, which was based on deep convolutional network trained on Graphic Processing Units (GPUs), see [18]. After such an event, ANN approaches to deep learning have been used in lot of different areas such, e.g., autonomic car driving, robot aided surgery, guidance of *intelligent drones*, software development, nuclear physics, and finance. Concerning the financial framework, different type of models related to the problem of predicting assets prices movements as well as the behaviour in time of more structured financial instruments, have been considered. In most of the cases, proposed approaches have been mainly based on statistical models used to analyse time series of interest, see, e.g., [12, 13, 14] and references therein, or by exploiting stochastic calculus techniques, see, e.g., [7, 8], and references therein, or by considering fine tuned *expansions techniques*, as, e.g., in [11, 9], see also [20], and references therein, while we refer to [5, 6], and references therein, for optimization problems related to the dividend and investment policy of a firm under debt constraints, resp. to portfolio liquidation startegies for large portfolio position in a limit-order market, and to [3], and references therein, for models related to the spread of contagion in financial networks. Is it worth to mention that, for what concerns the *machine learning approaches*, they are widely applied also to credit scoring, portfolio management, algorithmic trading, see, [22, 24, 10], respectively. In the present work we are dealing with the ANN-machine learning approach to the study of stock price movements prediction, a theme of increasing relevance in actual financial markets, particularly from the point of view of the so called *fast trading* management of order books.

The paper is structured as follows: in Sec. 2 we provide a short review of known techniques used for the stock price prediction and mainly based on *classic* machine learning algorithms, logistic regression or support vector machine; in Sec. 3, we consider more specifically the *Multilayer perceptron*, the *Convolutional neural network*, and the *Recurrent neural network* architectures; then; in Sec. 4, we discuss input data for algorithms, feature selection and preprocessing; eventually, in Sec. 5, we provide

the computational results related to the training processes, based on S&P500 time series; in Sec. 6 we take into consideration optimized selection procedures for the(hyper)parameters characterizing our architectures, focusing the attention on the use of the *Tree-structured Parzen Estimator* (sequential model-based) optimization scheme; in Sec. 7 we consider possible alternatives for our the study of the considered S&P500 time series from the machine learning point of view, namely the Principal Component Analysis, the feature selection approach and the use of *ensembles of neural networks*; eventually, in Sec. 8, we provide a numerical as well as a graphical interpretation of the results obtained exploiting the aforementioned approaches, and then commenting them in Sec. 9.

2 Existing approaches

The approaches used to forecast future directions of share market prices are historically splitted into two main categories: those that rely on technical analysis, and those that rely on fundamental analysis. Technical analysis uses only historical data such as, e.g., past prices, volume of trading, volatility, etc., while fundamental analysis is based on external information like, e.g., interest rates, prices and returns of other assets, and other macro/micro-economic parameters.

The *machine learning* approach to the latter problem has been declined in several forms, especially during recent years, also aiming to find an effective way to predict sudden financial crashes as, e.g., the one happened during the 2007-08 biennium. As an example, good results have been obtained using linear classifiers as the logistic regression one, which has been used to predict the Indian Stock market, see [15], or with respect to the S&P500 index, see [4]. More complicated techniques, as large-margin classifier or Support Vector Machine (SVM), was the best choice for prediction before the rise of neural networks. The latter uses a *kernel trick* which allows to consider our input data as embedded into a higher-dimensional space, where it is linearly separable. Successful applications of SVM have been developed, e.g., with respect to the analysis the *Korea composite* stock prices market, see [26], and the NIKKEI 225 index, see [16], clearly showing how SVM outperforms logistic regression, linear discriminant analysis and even some neural network approaches. Another popular technique based on decision trees, i.e. the *random forest* one, was also applied to similar problems as in, e.g., [19], where the authors studied the BM&F/BOVESPA stock market, resulting in a high accuracy obtained by using as input features different technical indicators such, e.g., simple and exponential moving averages, rate of change, stochastic oscillator, etc. The aforementioned references are all characterized by an accuracy in stock price movement forecasting that ranges between 70% and 90%. In what follows, we show that is possible to achieve the same accuracy by mean of neural networks, working with preprocessed open/close/high/low data, also working with high frequent, intra-day, data.

3 Artificial neural networks

3.1 Multilayer perceptron

Artificial Neural Networks look like electronic models based on the neural structure of the brain, which basically learns from experience. The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes, while the inputs are fed directly to the outputs via a series of weights. In this way, it can be considered the simplest kind of feed-forward network. Multilayer neural network consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function. The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds for a wide range of activation functions as in the case of sigmoidal functions.

3.2 Convolutional neural network

Convolutional neural network is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field. There are several different theory about how to precisely define such a model, but all of the various implementations can be loosely described as involving the following process:

- convolve several small filters on the input image;
- subsample this space of filter activations;
- repeat steps 1 and 2 until your left with sufficiently high level features;

- use a standard a standard MLP to solve a particular task, using the results features as input.

The main advantage of convolutional neural networks is that we use convolutional and downsampling layers as learnable feature extractor, which allows us to feed neural network without sophisticated preprocessing, so that useful features will be learnt during the training. In our case, instead of considering the standard framework of 2D images, we apply convolutional network to 1D data, namely to (financial) time series.

3.3 Recurrent neural network

A recurrent neural network (RRN) is any artificial neural network whose neurons send feedback signals to each other. The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks this is not the better idea. In particular, if one wants to predict the next word in a sentence, then it is better if he knows which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a *memory* which captures information about what has been calculated so far. It is worth to mention that, eve in RNNs are theoretically able to treat arbitrarily long sequences of information, in practice they are limited to looking back only for few steps.

4 Data processing

In what follows we focus our attention on the S&P500 index, exploiting related data from 1950 to 2016 (16706 data points). Usually stock market data looks like on Figure 1 and 2, which report the *close prices* for every days in the aforementioned time interval (Fig.1).

Our goal is to predict the movements of the S&P500 index, exploiting some information from previous data. Suppose we are going to predict if the close price of the next day, resp. minute, is larger or smaller than the previous one, based on the last 30 days, resp. minutes, of observations. Appropriate time window and prediction horizon should be chosen during hyper parameter optimization stage. First we have to scale or normalize our input data. The input vectors of the training data are normalized in such a way that all the features have zero-mean, and unitary variance. Usually data are scaled to be in the range [-1; 1], or [0; 1]. In the neural network approaches

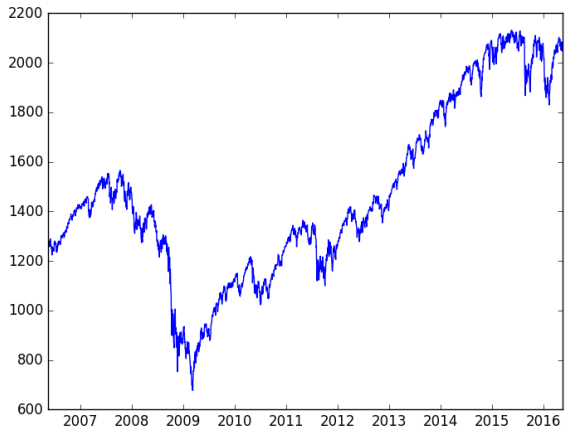

Figure 1: S&P500 index data from 2006 to 2016.

such renormalization strongly depends on the activation function of every *neuron*. A different approach, see, e.g., [7], is about considering not raw open or close ticks, but calculating return during the day, resp. during the minute, and then using this data as the training set. In Fig. 2, we have reported the plot of returns for the S&P500 index.

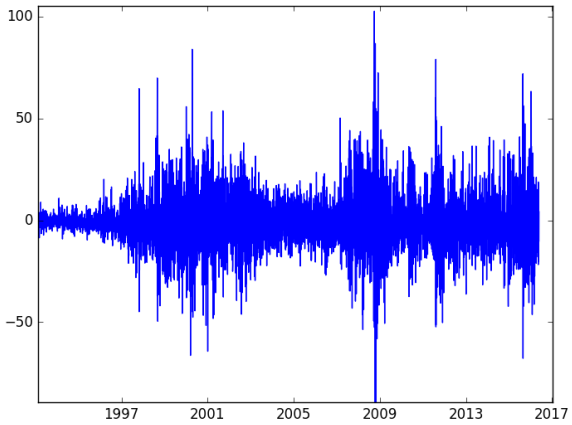

Figure 2: S&P500 index daily returns data.

In our research we use return prices as more representative data with normalization for stock price movement forecasting problem. In particular, we normalized our time series to have zero mean and unit variance, making use of the *sklearn library*, see [23]. All existing time series have been splitted, accordingly to the latter, in train, 80%, resp. in test dataset, the remaining 20%; moreover we use 10% of training set for hyper parameter optimization. Every element of the train data set is a normalized time series with

length of 30. Based on such a subdivision, we want to predict the transition [1; 0] if price goes up, resp. [0; 1] if it goes down, next day, resp. minute, according with the particular index chosen.

4.1 Preprocessing for regression problem

A classical approach in time series forecasting is represented by regression, predicting the exact value in the future based on given known historical time window. Neural networks can face this problem as well. The main difference from the point of view of the classification architecture, consists in changing the last affine layer which turns to predict real value. It follows that the activation function should be a linear function within the range -inf, +inf, moreover only one neuron should be present in this layer, instead of number of classes.

There is an important issue affecting the data preparation stage before regression. In fact, we need first to normalize data to be in some fixed range, them, after the regression procedure ends, we have to restore obtained results to predict actual value. We can suppose that both the mean and the standard deviation from historical time window are the same as for the next value, so that, for training data, we first normalize every time window and then subtract its mean and divide for its standard deviation, in single prediction value. We point out that a common mistake consists in considering the the mean and the standard deviation for target as a part of time window, affecting a lot the final results.

5 Experimental results

In this section we provide the computational results related to the training processes. All NNs were trained using the Keras deep learning framework for Python. Every NNs for S&P500 data was trained for 10 epochs, exploiting the Adam optimization algorithm, see [17]. Computational time was reduced by using GPU hardware, in particular we made use of a GTX 860M to speed up tensor multiplication, as well as other mathematical routines. In the hyperparameters optimization section, see Sec. 6, we tune all the parameters, e.g., the number of hidden layers, neurons, optimization algorithm, etc., with respect to architectures.

5.1 Multilayer perceptron architecture

We use the Multilayer perceptron architecture (MLP) with two hidden layers and test different layer sizes to determine the optimal size from the point of view of data modelling. We choose the rectified linear unit (ReLU) function as activation function, and, between two hidden layers, one dropout layer is included to prevent over fitting. The architecture of this model is shown in Fig. 3.

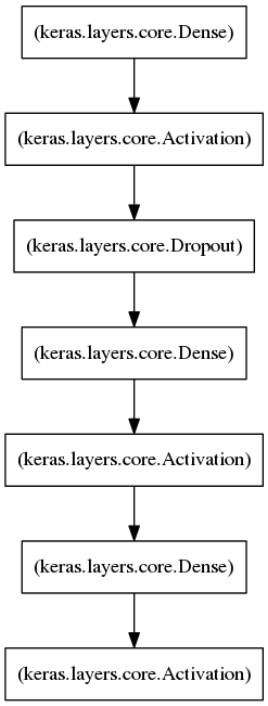

Figure 3: MLP architecture.

5.2 Convolutional neural network architecture

We use the Convolutional neural network architecture (CNN), as a sequential combination of 1D convolutional and max-pooling layers, choosing hyper parameters as follows:

- Number of filters = 64;
- Pool length = 2;
- Subsample length = 1;
- Activation function ReLU.

We provide experiment results with 1 and 2 hidden convolutional layers. The architecture of this model is shown in Fig. 4.

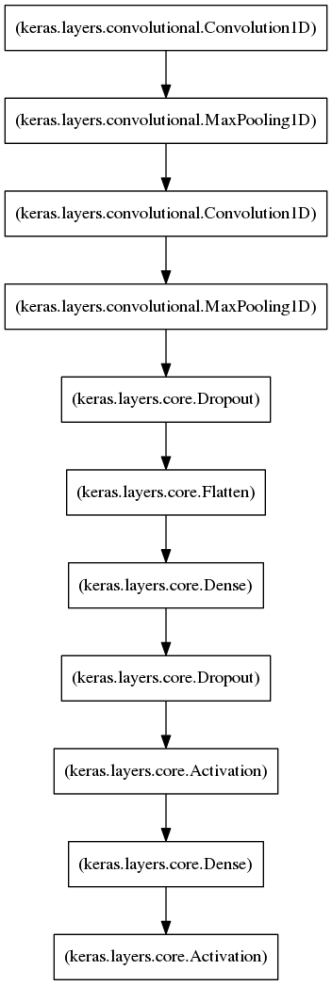

Figure 4: CNN architecture.

5.3 Recurrent neural network architecture

As a particular type of Recurrent neural network architecture (RNN), we choose the Long Short-Term Memory (LSTM) architecture, with 2 stacked recurrent layers and a *softmax* in the end. The number of hidden neurons equals 100, while the activation functions are hyperbolic tangent, while the inner activations are hard sigmoid functions. The architecture associated to the latter model is shown in Fig. 5.

5.4 Experimental Wavelet + CNN

One of the non-traditional approaches for financial time series prediction with NNs is the so called Wavelet-NNs (WNN). Basically, it feature extraction is based in wavelet transform, taking details part and skipping trend, which are then elaborated exploting a MLP, see, e.g., in [25]. Since CNNs showed better convergence and accuracy in previous experiments, hence we decided to pass time series preprocessed with Haar wavelets to the 2-layer CNN.

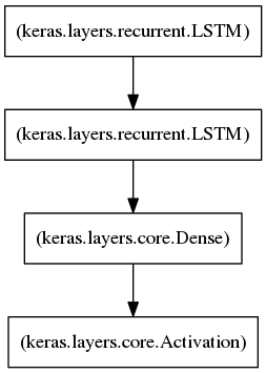

Figure 5: RNN architecture.

6 Hyperparameters optimization

We need to select the best parameters for our architectures such as number of neurons, hidden layers, dropout ratio, optimization algorithm, and the number of filters in CNNs. It can be done by hyperparameters search, namely by using the following main techniques: grid search, random search and the tree-structured Parzen Estimator.

The Tree-structured Parzen Estimator (TPE), see [1], is a sequential model-based optimization (SMBO) approach. SMBO methods sequentially construct models to approximate the performance of hyperparameters based on historical measurements, and then subsequently choose new hyperparameters to test based on this model. The TPE approach models $P(x|y)P(x|y)$ and $P(y)P(y)$, where x represents the hyperparameters, while y is the associated quality score. In particular, $P(x|y)P(x|y)$ is modeled by transforming the generative process of hyperparameters by replacing the distributions of the configuration prior with non-parametric densities.

The main steps of the aforementioned procedure are the following:

- describe the search space, that is hyperparameters and possible values for them;
- implement the function to minimize, which is nothing but our neural network model;
- optionally, take care of logging values and scores for analysis.

The function to minimize takes hyperparameters values as input and returns a score, that is a value for error, since we are minimizing. The latter means that each time the optimizer decides on which parameter values it likes to check, we train the model and predict targets for validation set, or we perform a cross-validation. Then we compute the prediction error and give it back to the optimizer. After the latter step, it again decides which values to check, and the cycle

starts over. In particular, we consider the following set of parameters

6.1 The set of parameters for MLP

- N of hidden layers - choice from [2, 3];
- N of neurons in every layer [64, 126, 250, 512];
- Values of dropout rates - uniformly distributed - 0, 0.5;
- Optimizing algorithm - choice from Adam, AdaDelta and RMSProp.

After 100 iterations of TPE was chosen model with two hidden layers, first with 500 neurons, second with 250. Dropout values were chosen quite high - 0.75 and 0.6 for both layers. Best optimizing algorithm is Adam.

6.2 The set of parameters for CNN

- N of convolution filters - choice from [32, 64]
- Filter length - choice from [2, 3]
- Pool length - choice from [2, 3]
- N neurons in fully-connected layer - choice from [100, 250]
- Value of dropout rate - uniformly distributed - 0, 1
- Optimizing algorithm - choice from Adam, AdaDelta and RMSProp

After 100 iterations of TPE, a two hidden layers model has been selected, with filter length of 2, pool length of 2, and with 64 filters in only one convolutional layer. Dense layer has to have 250 neurons and optimization algorithm has to be AdaDelta.

6.3 The set of parameters for RNN

- N of cells in every recurrent layer - choice from [50, 100];
- Optimizing algorithm - choice from Adam, AdaDelta and RMSProp.

After 50 iterations of TPE, a two hidden layers model has been selected, with 100 neurons each, optimized by Adam.

7 Further experiments

7.1 Principal components analysis

The Principal Components Analysis (PCA), see, e.g., [21], can be seen as an algorithm for linear dimensionality reduction. It is mainly based on the Singular Value Decomposition (SVD) technique for time series. SVD-type approaches aim at keeping only the most significant singular vectors to project the data to a lower dimensional space. PCA is mostly used as a tool in exploratory data analysis and for making predictive models. The PCA approach implies an eigenvalue decomposition of a data covariance (or correlation) matrix, or a singular value decomposition of a data matrix, usually after mean centring, and normalizing or using Z-scores. We would like to underline that such an approach is useful to reduce the number of variables when using algorithms that do not rely on time nature of input data, such as feed-forward neural networks. In our analysis we have chosen the 16 most relevant variables by exploiting the PCA technique, which allows us to achieve the results reported in subsection. 8.2.3

7.2 Feature selection

Another classical approach in machine learning is the so called *feature selection*, which can be used for the selection of feature as well as for dimensionality reduction on sample sets, or even to improve the estimators accuracy scores, and to boost their performance on very high-dimensional datasets. There are several variants on how to use it:

- Variance threshold: this is a simple baseline approach to feature selection. It removes all features whose variance does not meet some threshold. By default, it removes all zero-variance features, namely those features that have the same value in all samples.
- Univariate feature selection: such an approach works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator.
- Recursive feature elimination (RFE): this approach starts working on a given external estimator that assigns weights to features, e.g., the coefficients of a linear model, then the RFE consists in selecting features by recursively considering smaller and smaller sets of them. First, the estimator is trained on the initial set of features and weights are assigned to each one of them. Then the features whose absolute weights are the smallest, are pruned from the current set of features. The next step consists in recursively repeat the same procedure on the pruned set, until the desired number of features to select is eventually reached.
- A particularly effective procedure is constituted by implementing a *feature selector*, by exploiting some trained model. As an example, we can use random

trees algorithms which are very robust and precise in classification tasks. In particular such an implementation allow us to compute feature importances, which in turn can be used to discard irrelevant features.

We have used the last option, choosing important features from AdaBoost algorithm, from sklearn library.

7.3 Ensembles of neural networks

In machine learning, particularly in the creation of artificial neural networks, ensemble averaging is the process of creating multiple models and combining them to produce a desired output, as opposed to creating just one model. Frequently an ensemble of models performs better than any individual one, mainly because the various errors of the models *average out*.

In particular, we use the following strategy: first we create a set of experts with low bias and high variance, and then we average them. Latter approach implies the creation of a set of experts with varying parameters which are frequently represented by the initial synaptic weights. Nevertheless, we would like to underline that other factors, such as the learning rate, momentum etc., may be varied as well. Summing up, we implement the following steps:

- generate N experts, each with their own initial values;
- train each expert separately;
- combine the experts and average their values.

It is worth to mention that it is also possible to consider a combination of two or more models in order to create a *new one*, which performs better than any of them. In fact, since different models have different weak and strong sides, ad hoc *blending procedure* of them, may significantly improve the performance obtained by each single component. In order to concretely realize such an approach, the key idea we use is to learn weights of model using some simple classifier as, e.g., logistic regression or another neural network.

We used ensembles of three types: average, weighted and blending. We chose the best models from previous experiments in order to achieve better results using their ensemble. As blending top classifier we have used a regular logistic regression to get the weights of simple predictors.

8 Results

8.1 Neural networks for regression

Using the architectures described in previous sections, we have faced our problem as a regression problem. In figures 6, 7 and 8, we report a graphical visualization of obtained predictions

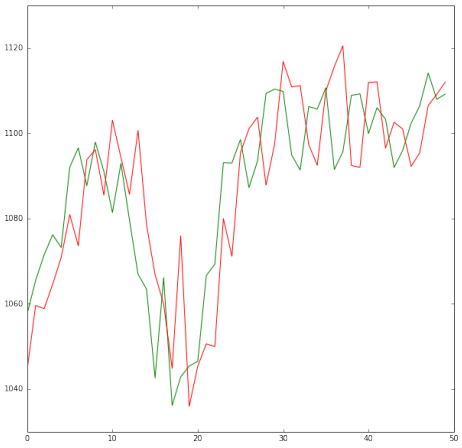

Figure 6: MLP architecture. Green line - actual value, red line - predicted

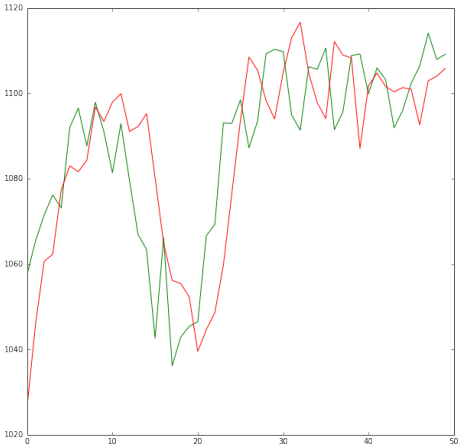

Figure 7: CNN architecture. Green line - actual value, red line - predicted

Looking at the MLP,CCN and RRN graphs, we clearly observe that predicted values are *behind* the actual data, which implies that neural networks can follow the trend, but cannot predict the *exact* future values of interest.

8.2 Neural networks for classification

8.2.1 Raw features

In Fig. 15 we have displayed the results related to the training of different architectures on daily returns. It

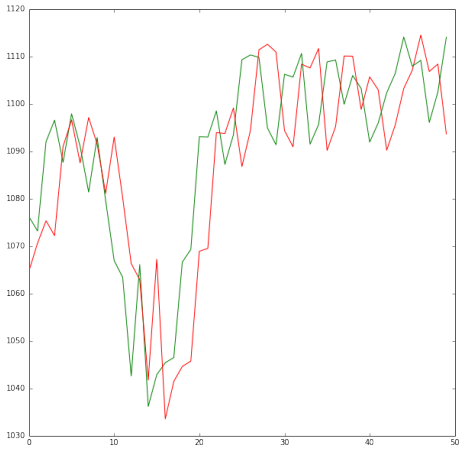

Figure 8: RNN architecture. Green line - actual value, red line - predicted

is easy to see that, from this perspective, CNNs are the best choice for modelling financial time series. Such a result suggests us to use CNN as basic predictor. Plots of accuracy and losses changing during training, are reported in figures from 9 to 14. In particular such graphs show that, due to the stochastic nature of data we are working with, the training procedure performs slowly and messy.

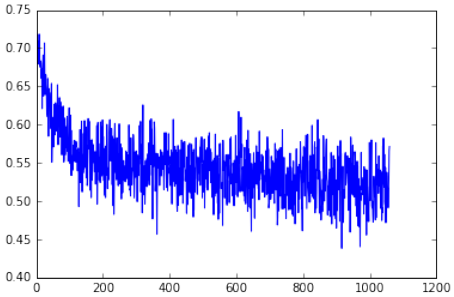

Figure 9: MLP loss function within 10 epochs

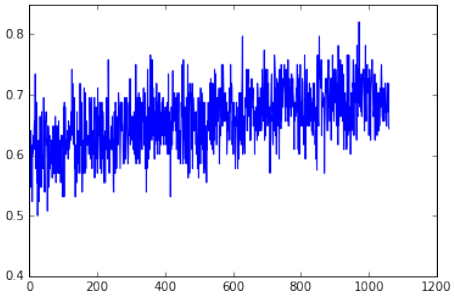

Figure 10: MLP accuracy within 10 epochs

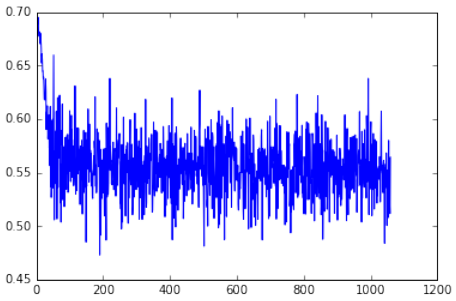

Figure 11: CNN loss function within 10 epochs

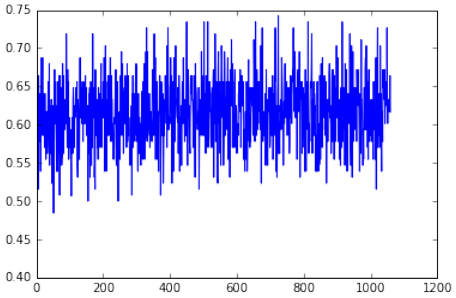

Figure 12: CNN accuracy within 10 epochs

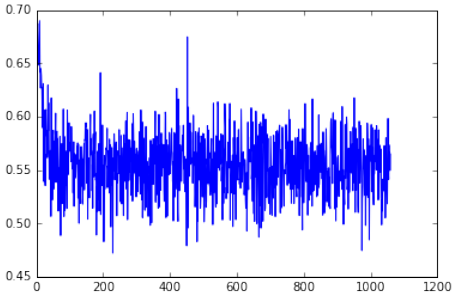

Figure 13: RNN loss function within 10 epochs

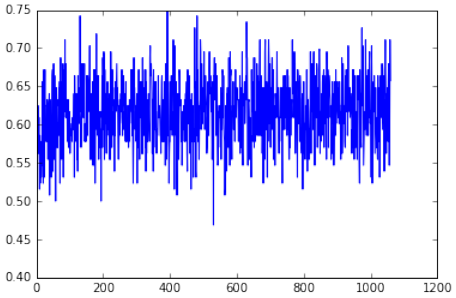

Figure 14: RNN accuracy within 10 epochs

8.2.2 Wavelet-CNN

After training CNN on wavelet decomposed data, with respect to the details of a signal, we obtain: MSE = 0.24, with an accuracy value equal to 0.55, hence obtaining a slight increase of performance if compared

	MSE	Accuracy
MLP	0.26	0.521
CNN	0.2491	0.536
RNN	0.2498	0.522

Figure 15: MSE and accuracy after training different architectures

	MSE	Accuracy
Average ensemble	0.23	0.558
Hand-weighted ensemble	0.23	0.56
Blended ensemble	0.226	0.569

Figure 16: MSE and accuracy after training different ensembles

with previous results based on raw data.

8.2.3 PCA selected features

We would like to point out that the application of CNN to PCA selected features, turns to produce poor results. In fact, by implementing such an approach we lose the time-based structure of data. Then we trained MLP on PCA-transformed data, achieving MSE = 0.26, with an accuracy value of 0.51, hence a worse result, even if compared with raw trained networks. Latter fact is not surprising, because in this case time series is ruined by having removed some data points.

8.2.4 Tree-based selected features

We can train CNN on data from tree-selected features, in order to choose CNN as a classifier, because random trees chose last data points as significant data for classification, hence without corrupt the time series structure. Applying this method we achieve: MSE = 0.256, with an accuracy value of 0.525, hence a result at least comparable to the one obtained by the MLP approach.

8.2.5 Ensembles of neural networks

Exploiting the results reported along previous sections, we have chosen the CNN trained on raw features model, the W-CNN model and the RNN model, to create an ensemble which performs better than its single components. We have reported in Fig. 16 the results obtained applying average, hand-weighted ensemble and blending.

As hand weights, we have empirically chosen 0.3 for CNN, 0.25 for RNN and 0.45 for W-CNN, as the most accurate predictor.

After learning blending with logistic regression we realized that it was hypothesis - learned (and normalized) weights were 0.32 for CNN, 0.19 for RNN and 0.49 for W-CNN, which shows us, that result of W-CNN is the most important for an ensemble.

9 Conclusions

In this work different artificial neural network approaches, namely MLP, CNN, and RNN, have been applied to the forecasting of stock market price movements. We compared results trained on a daily basis, for the S&P500 index, showing, in particular, that *convolutional neural networks* (CNN) can model financial time series better then all the other considered architectures.

We also implemented a novel Wavelet + CNN algorithm which outperforms other neural network approaches. In particular, it shows that feature preprocessing is one of the most crucial parts in stock price forecasting.

We would like to underline that we achieved much better results using ensembles of neural networks. It is worth to mention that such an approach has showed its power particularly when dealing with large-scale classification tasks, such, e.g., Netflix customer classification, see [2].

Our ongoing researches aim at treating more extended financial time series, and their features, in order to improve the training performance as well as related accuracy, for the particular type of neural networks considered. Techincal indicators, moving averages and stochastic oscillators, will be also taken into account to increase accuracy of prediction.

References:

[1] J.Bergstra et al. Algorithms for Hyper-Parameter Optimization *NIPS 2015*

[2] James Bennett, The Netflix Prize *Proceedings of KDD Cup and Workshop*, 2007

[3] Benazzoli, C., Di Persio, L. , Default contagion in financial networks *International Journal of Mathematics and Computers in Simulation*, 10, pp. 112-117, 2016

[4] S.-H.Chen, Genetic Algorithms and Genetic Programming in Computational Finance, Springer, 2002

[5] Chevalier, E., Vath, V.L., Scotti, S., An optimal dividend and investment control problem under debt constraints *SIAM Journal on Financial Mathematics*, 4 (1), pp. 297-326, 2013

[6] Chevalier, E., Vath, V.L., Scotti, S., Roch, A. Optimal execution cost for liquidation through a limit order market *International Journal of Theoretical and Applied Finance*, 2016

[7] Cordoni, F. and Di Persio, L., Backward stochastic differential equations approach to hedging, option pricing, and insurance problems (2014) International Journal of Stochastic Analysis, 2014.

[8] Cordoni, F. and Di Persio, L., Invariant measure for the Vasicek interest rate model in the Heath-Jarrow-Morton-Musiela framework (2015) Infinite Dimensional Analysis, Quantum Probability and Related Topics, 18 (3).

[9] Cordoni, F. and Di Persio, L., First order correction for the characteristic function of a multidimensional and multiscale stochastic volatility model (2014) International Journal of Pure and Applied Mathematics, 93 (5), pp. 741-752.

[10] Cont, R., Kukanov, A. Optimal order placement in limit order markets *Quantitative Finance*, pp. 1-19, 2016

[11] Di Persio, L., Pellegrini, G. and Bonollo, M., Polynomial chaos expansion approach to interest rate models, (2015) Journal of Probability and Statistics.

[12] Di Persio, L. and Frigo, M., Gibbs sampling approach to regime switching analysis of financial time series (2016) Journal of Computational and Applied Mathematics, 300, pp. 43-55.

[13] Di Persio, L. and Frigo, M., Maximum likelihood approach to markov switching models (2015) WSEAS Transactions on Business and Economics, 12, pp. 239-242.

[14] Di Persio, L. and Perin, I., An ambit stochastic approach to pricing electricity forward contracts: The case of the German Energy Market (2015) Journal of Probability and Statistics.

[15] A. Dutta, G. Bandopadhyay and S. Sengupta, Prediction of Stock Performance in the Indian Stock Market Using Logistic Regression, *International Journal of Business and Information*, 2012

[16] Wei Huang, Yoshiteru Nakamori, Shou-Yang Wang, Forecasting stock market movement direction with support vector machine, *Computers and Operations Research, archive Volume 32, Issue 10*, 2005, pp. 2513–2522

[17] D. P. Kingma, J. Lei Ba, Adam: A method for stochastic optimization, *3rd Int. Conf. for Learning Representations, San Diego*, 2015

[18] A. Krizhevsky, I. Sutskever and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, *NIPS 2012, Nevada*, 2012

[19] Marcelo S. Lauretto, B. C. Silva and P. M. Andrade, Evaluation of a Supervised Learning Approach for Stock Market Operations, *arXiv:1301.4944[stat.ML]*, 2013

[20] Marinelli, C., Di Persio and L., Ziglio, G., Approximation and convergence of solutions to semilinear stochastic evolution equations with jumps (2013) Journal of Functional Analysis, 264 (12), pp. 2784-2816.

[21] Pearson, K. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine 2 ,pp. 559572.* , 1901

[22] Rodan, A., Faris, H. Credit risk evaluation using cycle reservoir neural networks with support vector machines readout *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* , 9621, pp. 595-604, 2016

[23] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research Volume 12*, 2011, pp.2825–2830

[24] Novak, G. Veluscek, D. Prediction of stock price movement based on daily high prices *Quantitative Finance*, 16 (5), pp. 793-826, 2016

[25] Chong Tan, Financial Time Series Forecasting Using Improved Wavelet Neural Network, *Master Thesis, University of Aarhus*, 2009

[26] Y. Wang and In-Chan Choi, Market Index and Stock Price Direction Prediction using Machine Learning Techniques: An empirical study on the KOSPI and HSI, *Technical report*, 2013

[27] V.V.Kondratenko and Yu. A Kuperin, Using Recurrent Neural Networks To Forecasting of Forex, *arXiv:cond-mat/0304469 [cond-mat.dis-nn]*, 2003