Group Project - Intro to Deep Learning

Valentin Courgeau 17098662 valentin.courgeau.17@ucl.ac.uk [*]
Giang Dang 17055511 g.dang.17@ucl.ac.uk [†]
John Goodacre 13064947 john.goodacre.13@ucl.ac.uk [*]
Pier Francesco Procacci 17075285 pier.procacci.17@ucl.ac.uk [*]

December 2017

1 Introduction

This project originates from an interest in financial timeseries as most group members have a background related to financial data. We were motivated by the application of the material covered in the lectures to tackle those series known to be noisy and difficult to deal with.
We were keen on exploring both classification and prediction perspectives: we devote the first part of this report to a specific classification implementation [**Add reference to original paper**] whilst the second part will cover a prediction-based implementation from Bao et al. [2].
More precisely, we will be interested in a financial index, the S&P500, based on 500 large American companies and generally taken as a representative number of the American economy's performance. The first approach will carry out a classification approach on individual constituents of the S&P500 compared to the average performance of the market. It will involve the use of LSTM and Momentum effect as described below [**Add details**]. The second part will be based on prediction of the index itself accompanied with so-called *technical indicators* supposed to give information on the trends, selling/buying pressure, etc of the index. This comprises the use of denoising modules such as Wavelet Transform, Stack Autoencoders and stacked LSTM's.

2 A classification approach

2.1 Purpose

The Momentum Effect is, in finance, the empirically observed tendency for stock prices to continue their trend: rising asset prices to rise further, and falling prices to keep falling. This is effect is well known among financial practitioners and, based on this effect, a number of trading strategies have been developed. In literature, a first notable claim in favour of trading strategies based on momentum effect is due to Levy (1967) who defined these strategies as "Relative Strength" strategies, before becoming popular even among academics during '90s after the contributions of Jegadeesh and Titman (1993) and Carhart (1997).

Jegadeesh and Titman (1993) introduced what is considered today a standard definition of momentum based strategies: "At the beginning of each month t the securities are ranked in ascending order on the basis of their returns in the past J months, excluding the previous one. Based on these rankings, ten decile portfolios are formed that equally weight the stocks contained in the top decile, the second decile and so on. The top decile portfolio is called the 'losers' portfolio and the bottom decile is called the 'winners' portfolio".[1]
Inspired by Takeuchi and Lee (2013), in this section we aim to ... describe structure

[*]MRes - CDT in Financial Computing and Analytics
[†]MSc Data Science & Machine Learning
[1]Jegadeesh N., Titman S., *Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency*, The Journal of Finance Vol XLVIII, March 1993

2.2 Data

We thank the Department of Computer Science at University College London and Bloomberg for data on individual US stocks.

2.2.1 Dataset specification

We restrict our analysis to the constituents of S&P 500 Index from January 1990 and December 2016. These are 227 stocks and we considered weekly prices of ordinary shares.
The training set covers the period from January 1990 to December 2008 and contains 800 examples for each individual stock, for a total of 181,600 weekly examples. The test set covers the period from January 2008 to December 2016 and contains 405 examples for each stock, for a total of 91,935 weekly examples.

2.2.2 Preprocessing

In providing inputs to our model, the objective is twofold: limiting hand-engineered features from historical data and, at the same time, obtaining a dataset coherent with a momentum approach. For every week t, we considered the 52 weekly returns from t-5 to t-16. This means considering the returns in the past J=12 months excluding the previous one, coherently with the standard approach by Jegadeesh and Titman (1993). Our input training set is, therefore, constituted by a (52 returns x M stocks x N periods) tensor of overlapping windows as illustrated by Figure 2.

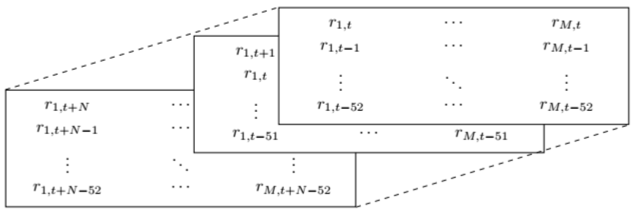

Figure 1: Dataset structure

While standard momentum trading strategies consider, for each stock, only a function of the returns over the considered time window (cumulative returns, average, ...) for the subsequent cross-sectional ranking, we aim to let the model extract the relevant features. Thus, over each time window we computed the series of 52 weekly cumulative returns for each stock and we normalize each of the cumulative returns by calculating the cross-sectional z-scores of all stocks for each week. This is due to both the need for normalization of inputs to be feed into a DNN and the fact that momentum is a cross-sectional effect. Figure 2, inspired by Takeuchi and Lee (2013), illustrate this transformation pipeline considering the 52 weekly returns for one stock in the training set.
Finally, we considered the returns over the t+4 week (subsequent month) as labels: returns above the cross-sectional median as belonging to class 1 and returns below the median as belonging to class 0.

3 A timeseries prediction approach

3.1 Purpose

We will now approach financial timeseries from a prediction perspective. We will provide a study of a model presented by Bao et al. [2] to predict one-day-ahead close quote of financial indices such as the S&P500, composed of 500 large American companies listed on the NYSE or NASDAQ. This model relies on denoising via Wavelet Transform, Stacked Autoencoders (SAEs) to reduce the dimensionality of the problem and stacked LSTMs to incorporate some temporal dependency. SAEs are also use to gain access to higher feature abstraction whilst LSTM are currently considered the state-of-the-art architecture when dealing with

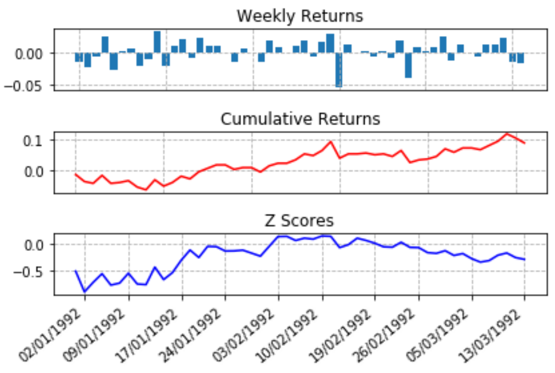

Figure 2: Preprocessing pipeline

timeseries as introduced in the seminal paper by Hochreiter and Schmidhuber [4] although we focus on its *deep* counterpart with stacked LSTM layers (Sak et al. [9]).

3.2 Data specification

The data used in the original paper [2] is the daily quotes of the S&P500 along with other technical indicators from 1st July 2008 until 30th September 2016. We listed the full list of indicators in the Appendix A, Table 2. This table was replicated from the original paper for completeness.
This means we work with 19 predictors to predict tomorrow's close price.

4 The model

4.1 Background

4.1.1 Discrete Wavelet Transform (DWT)

Idea Following [2], we used a Discrete Wavelet Transform on the each individual timeseries to remove some of the noise. The DWT is a linear transformation of a timeseries $(x(t_i))_{i=1}^N$ where $N = 2^J$ for J positive integer giving the number of multi-resolution scales. We project the original timeseries on the mother wavelet ψ and father wavelet ϕ. We define

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}}\psi(2^{-j}k) \quad \text{and} \quad \phi_{j,k}(t) = 2^{-\frac{j}{2}}\phi(2^{-j}k)$$

It decomposes the timeseries into low- and high-frequency components using low- and high-pass filters via respectively the father and mother wavelets. Its approximation can be written as follows:

$$x(t) = S_J(t) + D_J(t) + \cdots + D_1(t)$$

with S_J the coarsest approximation level and $D_j,\ 1 \leq j \leq J$, being the higher-frequency approximations. More precisely, we have:

$$S_J(t) = \sum_{k=1}^{2^J} s_{J,k}\phi_{J,k}(t) \quad \text{and} \quad D_j(t) = \sum_{k=1}^{2^j} d_{j,k}\psi_{j,k}(t) \quad \text{for } j = 1, \ldots, J$$

For more detailed information see the original paper [2], Section

Application The authors of [2] only provided a few details regarding the implementation and application of DWT. As advised in [5] and [2], we applied a 2-level decomposition twice, keeping only the *coarsest* approximation at each iteration with two levels (J = 2). Therefore, each decomposition creates three timeseries

S_2, D_2 and D_1 and we discard D_2 and D_1: this helps smooth the signal by only keeping the lowest frequency decomposition.
In practice, we used the piecewise-constant *Haar* mother wavelet $\psi(t) = \mathbb{1}_{[0;1/2[}(t) - \mathbb{1}_{[1/2;1[}(t)$ since Al Wadia et al. [1] concluded on the superiority of Haar's wavelet over Daubechies' wavelet in the forecasting. This means that the second application of the DWT does not bring any additonal smoothing, yet our implementation allows to quickly change the type of wavelet used. We relied on the **PyWavelets** package [10] and the 1D Multilevel DWT [WARNING] functionality. This implementation relies on Mallat's algorithm [8] for Fast Wavelet Transform computation.

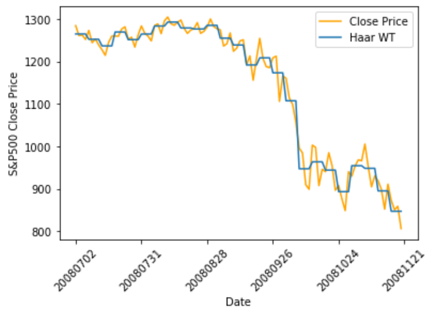

Figure 3: Comparing S&P500 Close Price and Haar wavelet transformed data on the first 100 dates

4.1.2 Stacked Autoencoders (SAE)

Autoencoders are a flavour of neural network whose architecture forces a lower dimensional representation of the input data. Conceptually, a basic autoencoder comprises an encoder and a decoder, each of which may have several layers. The encoder compresses the data into a latent representation. The decoder takes this latent representation and attempts to reconstruct the original data. Thus the target is the input, we minimise the error between the input and target using a function based upon the nature of the data being compressed. In our chosen paper [2], Bao et al chose a mean squared error with further regularisation on the norm of the weights of the neural net.

In its simplest form an autoencoder with a linear activation function is equivalent to Principal Components Analysis (optimised in an albeit inefficient manner). Fig 4 from the Keras blog, `https://blog.keras.io/building-autoencoders-in-keras.html` gives a visual intuition of a single autoencoder. The non-linearity of the activation function makes an autoencoder more versatile than PCA and enables non-linear dimensionality reduction.

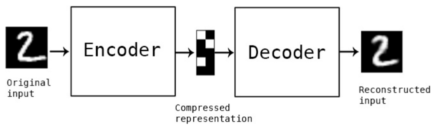

Figure 4: Simple Autoencoder

Stacked Autoencoders (SAE's) are autoencoders where the latent layer is itself further expanded as an autoencoder each additional time we stack. Conceptually the idea is to be able to learn more complex low dimensional feature representations of the original data. It is important to note that the implementation is not the same as a deep autoencoder. Bao et al [2] chose to use an SAE with 5 layers comprising 4 single layer autoencoders. The first autoencoder maps daily input variables into the first latent layer. The first decoder is then removed and the latent data is now used as the input layer to the second autoeconder, and so on.

The final latent data from their 4th stacked autoencoder was the final output of the SAE (subsequently used as inputs to the final stage of the overal model - stacked LSTMs). Figure 5, from [2] illustrates the exact architecture of the SAE part.

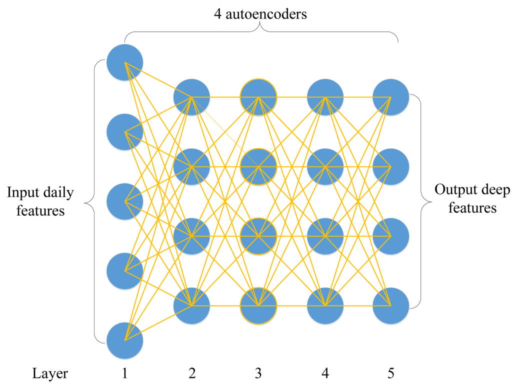

Figure 5: Stacked Autoencoder architecture as presented in [2]

The SAE was implemented in tensorflow with the following key components.

- 5 Layer Stacked Autoencoder.
- Bao et al [2] chose 10 dimensions for their hidden layers, which we replicated, but also generalised by enabling a flexibly implementation of stacking and dimensionality.

- Adam optimiser, learning rate 0.005 - Rationale - because it usually tends to work well out of the box.
- Loss function Mean Squared Error (following [2])
- L2 regularisation.
- tanh activation function (here we differ from [2] who used sigmoid). We encountered instability during some optimisations and found data normalisation combined with tanh gave more stable results.
- **Sanity Check** Financial data by its nature is very noisy with the potential for garbage in/ garbage out. Thus, to ensure the SAE was implemented correctly we first tested the architecture on the MNIST dataset. Figure 6, shows the SAE architecture (100 dimension hidden layers) applied to four unseen MNIST test images. As a pure sanity check, we examined the mean squared errors and added back the decoders in order to reconstruct the images, for eye-balling purposes.

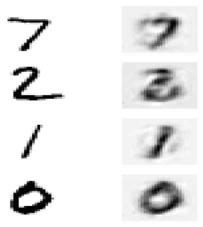

Figure 6: Quick sanity check, 5 layer stock market SAE applied to MNIST

Potential Criticism, autoencoders or stacked autoencoders are typically used for dimensionality reduction - Bao et al [2], appears to be one of the earliest adopters of the SAE applied to financial time series data. However, financial data by its nature is very noisy. Regardless of architecture, the quality of the latent output from the autoencoder is primarily depending upon the quality of the input data itself. Which is why we felt it necessary to test on known data such as MNIST first.

The input to the SAE in [2] was the output from wavelet transforms of the raw data. Thus the quality of the outputs from the SAE was co-dependent on a number of factors, the quality of input data itself, the ability of the wavelet transforms to smooth/ extract salient features and the architecture of the SAE. Because results in the world of finance can be inconclusive, we felt the need to prove further proof of concept by introducing a cheat feature (that is future information (with varying degrees of noise) added to our training data). The reasoning being to test in a more scientific manner the output from the whole architecture with features that we 'know' have predictive power.

4.1.3 LSTM's and Stacked LSTMs

LSTM's were first introduced by Hochreiter, Sepp and Schmidhuber, [4]. The key idea being to address a problem exhibited by standard recurrent neural networks, the difficulty of learning long term dependencies due to the vanishing gradient problem.

Fig 7 comes from the excellent blog by Colah `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, who describes the architecture in more detail.

The key to the architecture is the maintenance of a cell state, where information flow is regulated by three gates.

- **Forget Gate**: Takes the output from the previous time step, concatenates with input data from the current time step, multiplies by the forget gate's weights and applies a sigmoid activation. The output is multiplied by the previous cell state, thus 'forgetting' a certain proportion of the prior LSTM cell state, dependent upon the saliency of the input.

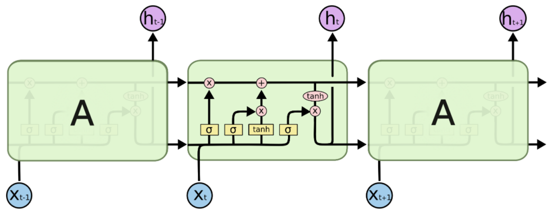

Figure 7: Illustration of a typical LSTM mechanism.

- **Input Gate**: Applies two transformations - one identical to the forget gate, but this time outputting a value deciding how much of the current data to 'input' to the current cell state. The second being a linear transformation with a tanh activation to create the new data to be proportionately added to the current cell state.
- **Output Gate**: Again applies a sigmoid transformation identical to the input and forget gates (again with its own set of weights), this is then multiplied by a tanh transformation of the current cell state - which is then outputted.

Stacked LSTMs are a series of LSTM's where outputs from one LSTM are fed as an input to the next. Clearly the output of the previous LSTM's must be sequences themselves. The intuition is similar to stacking autoencoders but in a time series sense, where one hopes to learn more complex hierarchical representations of the time series. The group's implementation of both papers utilised stacked LSTM's.

4.2 Architecture

Using the previously mentioned modules and the work of Bao et al. [2], we would like to study the one-day-ahead prediction of the S&P500. They proposed a three-step approach:

1. Smoothing: two consecutive 2-level DWT on each timeseries;
2. Encoding: SAEs to reduce the dimension from 19-dim down to 10-d;
3. Prediction: 4 10-d Stacked LSTMs with a lookback parameter;

[**Could add other details on the architecture**]

4.3 Training

As LSTM are often difficult to train, we opted for data normalisation (mean zero and standard deviation of 1 using only the training set) in addition to batch normalisation after each LSTM layer. We used *Adam* optimisation routine [6] as it is known to work well with LSTMs using the Mean Squared Error (MSE) loss function.
We used the traditional training/validation/test approach with proportions 80%/10%/10%. The first part is used to train the model, the validation set is used to tweak the hyperparameters and test set used to compare models with different architectures. The number of epochs is fixed to 1200 by trial and error using validation loss. The learning rate and decay are 0.002 and 10^{-5} whilst the SAE's and LSTM's batchsize is 60. Indeed, both training are performed consecutively.

4.4 Experiments Performed

We compare models using 3 standard metrics: MAPE, R and Theil U. Also, we used a simple buy-and-sell trading strategy as explained in B with buying/selling costs of 0.05%. Note that the original paper features a typo in their return formula which **adds** the transaction costs while in fact we have to pay them. *This test allows us to know if the model captures the patterns in the data whilst being more robust towards over-fitting since the model just has to guess to right move direction.* Finally, we were suspicious of the capacity of the model to retain useful information. To test that, we also include artificial test cases where the correct prediction was one of the input of the models. We denote those models as *cheat models* for obvious reasons.

Lookback parameter A key part of the model is the *lookback* parameter which determines how far we look back in the past in the LSTM modules. We noted that the behaviour of the stack of LSTM changed drastically when tweaking this parameter. With just the stack of LSTMs used (that is, without wavelet transform or SAE), we obtained the cumulative returns pictured as follows for lookbacks of 2, 3, 4 and 5: We observe a similar performance of cheat models (except with a lookback of 5). They also provide the best

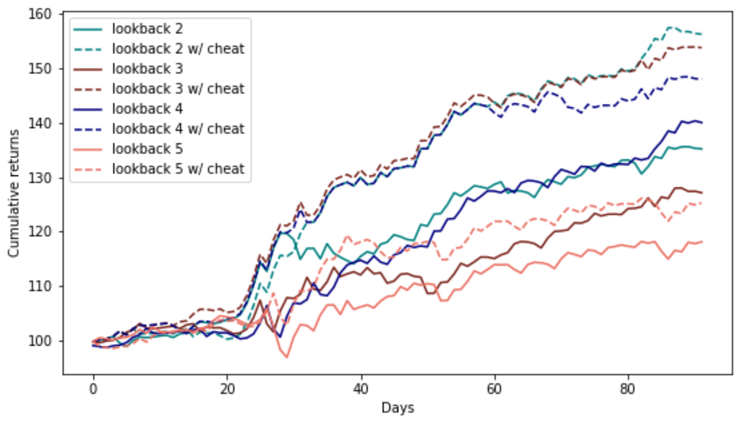

Figure 8: Cumulative returns indexed at 100 on the validation set for lookback parameters 2, 3, 4, 5. Dashed lines for cheat models, solid lines for original models.

performances with returns above 50%. Whilst the *lookback 2* and *lookback 4* models have similar returns, we will use the latter as it has the best overall returns among the original models. Also, for *lookback 2*, $MAPE = 3.43$, $R = 0.89$, $TheilU = 0.22$ and for *lookback 4*, $MAPE = 3.27$, $R = 0.91$, $TheilU = 0.24$ which indicates that the predictions are more accurate when using a lookback of 4 although the higher Theil U indicates that the predictions are more volatile for lookback 4.

Impact of the modules Wavelet transforms (as in [7]) and Stacked Autoencoders (as in [3]) are powerful denoising modules used respectively in timeseries analysis and computer vision. Their effectiveness has been proven numerous time but the consecutive use has not been seen in the context of financial timeseries prior to Bao et al. [2] hence why we were interested in testing their idea.
More precisely, we wanted to replicate a model comparison of their WT-SAE-LSTM model with sub-models constructed with/without certain modules. We also included *cheat models* as in the lookback parameter discussion above and an Multi-Layer Perceptron as a baseline.

Model	MAPE	R	Theil U	return(%)	Time(s)
MLP	0.0109	0.9438	0.0075	100.08	249.83
MLP w/ cheat	0.0016	0.9967	0.0019	200.00	238.73
LSTM lb 2	0.0183	0.8699	0.0112	10.18	598.23
LSTM lb 2 w/ cheat	0.0141	0.9008	0.0102	88.79	602.04
LSTM lb 4	0.0223	0.7559	0.0145	53.84	727.35
LSTM lb 4 w/ cheat	0.0071	0.9817	0.0042	114.83	558.31
WT-LSTM lb 4	0.0221	0.8129	0.0143	4.67	805.97
WT-LSTM lb 4 w/ cheat	0.0087	0.9725	0.0057	96.33	878.99
SAE-LSTM lb 4	0.0439	0.3917	0.0267	-24.47	892.20
SAE-LSTM lb 4 w/ cheat	0.0264	0.7523	0.0157	-9.49	1023.93
WT-SAE-LSTM lb 4	0.0273	0.6562	0.0169	7.14	1050.05
WT-SAE-LSTM lb 4 w/ cheat	0.0313	0.6751	0.0186	-5.90	892.19

Table 1: Results of prediction on the test set between various models. *lb = lookback*

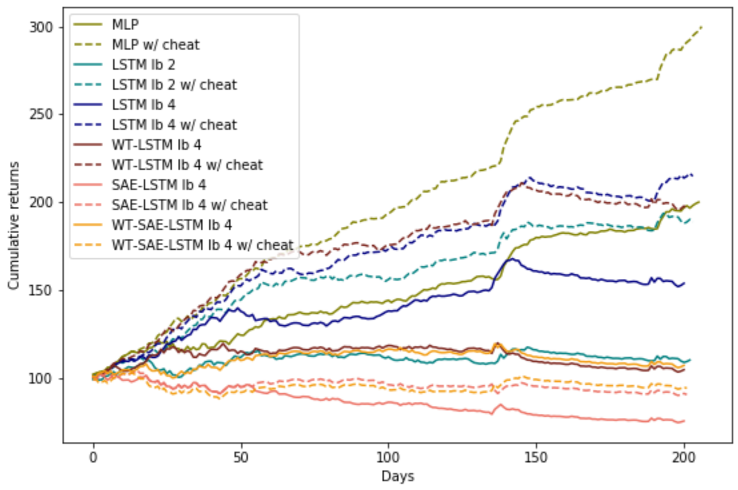

Figure 9: Cumulative returns indexed at 100 on the test set for lookback parameters 2 and 4 and with/without wavelet tranform (WT) and Stacked Autoencoders (SAE). Dashed lines for cheat models, solid lines for original models.

4.5 Further work, Criticisms

5 Results

5.1 Classification

The model was trained for each stock considering the examples from January 1990 to December 2008 and tested considering those from January 2008 to December 2016. This section refers to the performance obtained in the test set in terms of both prediction accuracy and financial performance.

5.1.1 Classification performance

The model achieves an overall accuracy rate of 51.936%. From a financial point of view, we consider the realized returns as a measure of model's performance. In order to compute the realized returns of the model, we construct a portfolio constituted by stocks predicted to be in class 1 as long positions (buy) and stocks predicted to be in class 0 as short positions (sell). This portfolio is then rebalanced weekly based on the new obtained predictions. This construction is consisted with a momentum trading strategy apart from the holding period. In a standard momentum strategy the portfolio constructed as described is, in general, supposed to be hold for K months, we K usually from 1 to 4. In our experiment, instead, we considered 1 week holding period. The average weekly return provided by the above described strategy is 0.11%.

5.1.2 Prediction and probabilities

While these results could be promising object of discussion and further work, we are not considering the whole deal of information provided by the model. The classification accuracy as well as the trading strategy returns described above use the 50% threshold in the probabilities to predict classes and to form long/short positions. Figure 10 shows the distribution of the estimated probabilities of being in class 1 for a stock in the test set. We see that there is a large portion of probabilities in the set [0.4,0.6]. Analysing the relationship among probabilities and holding period returns, we notice that the relationship is positive, meaning that a higher estimated probability of being in class 1 leads to a higher return.

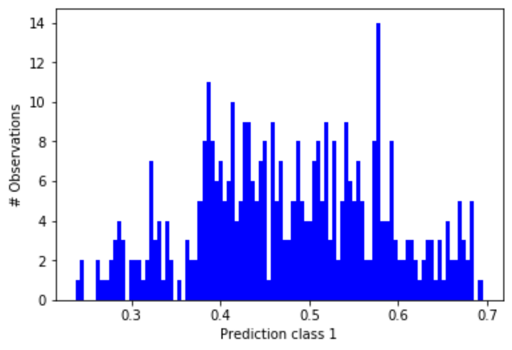

Figure 10: probabilities of being in class 1 for one stock in the test set

This observation suggests that we could exploit this feature in order to obtain better performances. Indeed, there is no requirement to hold all of the stocks in the dataset and, therefore, we can select only stocks for which the predictions are provided with high probabilities.

Each week, we considered for all stocks the predicted probabilities to be in class 1 and 0 and we we formed long positions (buy) with stocks predicted to be in class 1 with probability higher than 65& and short positions with those predicted to be in class 0 with a probability higher than 75%[2] Next week we close our positions forming a new long/short portfolio. Repeating this process over the test set generated a time

[2]Short positions are generally considered difficult to be forecasted, especially among large cap stocks as the S&P500 constituents. This is why we considered asymmetric thresholds.

series of weekly returns of the strategy. Figure 11 presents the cumulative returns of the strategy and of the considered benchmark index (S&P500) from January 2008 to December 2016.

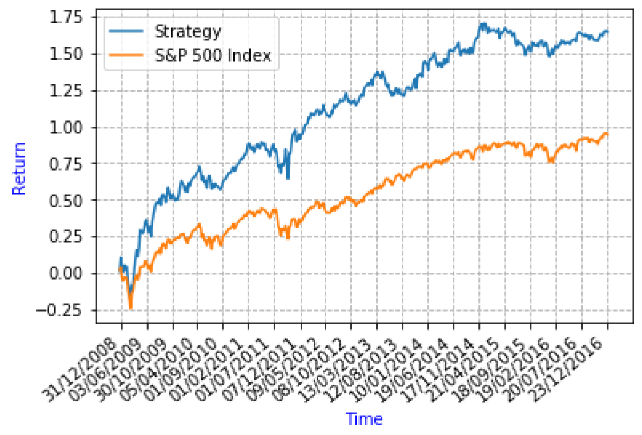

Figure 11: Cumulative returns proposed strategy vs benchmark

The proposed strategy generates an average weekly return of 0.41% as compared to a 0.23% average weekly return on the benchmark. On a cumulative basis, the proposed strategy outperforms the benchmark by 70% in 8 years.

5.1.3 Discussion

Predicting the financial world remains one of the most challenging problems in business community that researchers have been spending decades working on it. In this work, we implemented a neural network architecture in order to investigate the possibility of constructing a binary classifier upon the decision to buy or to sell the stocks. Even though the neural network could give high accuracy during training in our designated problem, it failed to obtain a significant outcome while exposing to the test set. Almost all stocks predicted from those models had the accuracy that fluctuates a little higher than 50% which is obviously not better than a random guess. Although we acknowledge that the back test from our suggested portfolio showed the model did have an effort to give some reasonable predictions, our main consideration for future development lies on how to improve the neural network performance applying in our case.

We list several possible reasons that could explain the behaviour of the model. Our first concern is the data itself. Given the chaotic nature in the financial world, it is not a trivial task to extract features from the noisy data. Secondly, given that the dimension of our dataset is exceptionally high, the number of data points is small in contrast. As specified in session 2, our dataset contains 800 examples for the training set and 405 examples for the test set. We believe that it is insufficient for the cumulative 1205 samples to learn the knowledge of 227 stocks with a 52 weekly returns timing window for each index. As treated in this problem we only considered the stocks that remained the S&P500 index from 1990 to 2016. The S&P500 index is periodically rebalanced in order to consider the largest 500 US stocks in terms of market cap and some stocks were delisted over the years. Going back further the year 1990 would imply a diminishing number of stocks and therefore would result a biased analysis as compared to the index.

Given the discussion on the factors that limited the model performance, we propose some suggestions for future work. Getting more data would be the first and foremost thing to be taken into account. As discussed the reason that prevented us to acquire more S&P500 indexes preceding the year 1990, it would be tricky to increase the size of our dataset. Reducing the dataset dimensionality could be another potential solution. The way we treated all 227 stocks as input to the networks might be re-considered in future work as some of these stocks are appeared to be unrelated to the others. For example, changes in tech giants such as Google or Apple might unlikely affect the rise and the fall in the stock indexes of other non-technological companies. However, the true relationships in each combination of the stocks remain unknown and in fact needs to be treated separately. This original setting of including all stocks to the input was our initial attempt to see if neural network models could help in exploring the interaction among the S&P500 index. In future work, we will focus on a smaller scale in which high correlated stocks would be grouped together and treated separately to see if reducing dimensionality under this scope could improve the network performance.

It is crucial to note that the above result was obtained given the outcome from the neural network. We presented a portfolio on the neural network's prediction and discussed the possible directions for future development of the project. Interestingly, it is suggested that one could make profit by following the buy/sell decision from our portfolio albeit being not fully convinced from the low accuracy neural network binary classification.

A List of technical factors

Table 2: Technical indicators presented in Bao et al. [2]

Name	Definition
	Daily Trading Data
Open/Close Price	Nominal daily open/close price
High/Low Price	Nominal daily highest/lowest price
Trading Volume	Daily trading volume
	Technical indicators
MACD	Moving Average Convergence Divergence; displays trend following characteristics and momentum characteristics.
CCI	Commodity channel index: helps to find the start and the end of a trend
ATR	Average true range: measures the volatility of price.
BOLL	Bollinger Band: provides a relative definition of high and low, which aids in rigorous pattern recognition.
EMA20	20 day Exponential Moving Average.
MA5/MA10	5/10 day Moving Average.
MTM6/MTM12	6/12 month Momentum: helps pinpoint the end of a decline or advance.
ROC	Price rate of change: shows the speed at which a stock's price is changing.
SMI	Stochastic Momentum Index: shows where the close price is relative to the midpoint of the same range.
WVAD	Williams's Variable Accumulation/Distribution: measures the buying and selling pressure. https://www.investopedia.com/terms/a/accumulationdistribution.asp-0
	Macroeconomic Variables
Exchange Rate	US Dollar Index
Interest Rate	Interbank Offered Rate; Rate at which banks can lend money to each other.

B Buy-and-Sell trading strategy

Denote y and y^* respectively the true and predicted close price timeseries. Given in [2], the *buy-and-sell* strategy unfolds as follows:

- If $y^*_{t+1} > y_t$ then we buy one unit of the underlying index and sell it the next day;
- If $y^*_{t+1} < y_t$ then we sell one unit of the underlying index and buy it the next day;
- If $y^*_{t+1} = y_t$ then do not buy/sell anything (this has never happened in our tests);

If we include buying/selling transactions (resp. a and b), the daily return is:

- If $y^*_{t+1} > y_t$, we obtain
$$\frac{y_{t+1} - y_t - (a * y_t + b * y_{t+1})}{y_t}$$
;

- If $y^*_{t+1} < y_t$, we obtain
$$\frac{y_t - y_{t+1} - (a * y_{t+1} + b * y_t)}{y_t}$$
;

References

[1] MTIS Al Wadia and M Tahir Ismail. Selecting wavelet transforms model in forecasting financial time series data based on arima model. *Applied Mathematical Sciences*, 5(7):315–326, 2011.

[2] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[3] Yushi Chen, Zhouhan Lin, Xing Zhao, Gang Wang, and Yanfeng Gu. Deep learning-based classification of hyperspectral data. *IEEE Journal of Selected topics in applied earth observations and remote sensing*, 7(6):2094–2107, 2014.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Tsung-Jung Hsieh, Hsiao-Fen Hsiao, and Wei-Chang Yeh. Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. *Applied soft computing*, 11(2):2510–2525, 2011.

[6] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] James B Ramsey. The contribution of wavelets to the analysis of economic and financial data. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 357(1760):2593–2606, 1999.

[8] Howard L Resnikoff and Raymond O Wells. The mallat algorithm. In *Wavelet Analysis*, pages 191–201. Springer, 1998.

[9] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[10] Filip Wasilewski and PyWavelets Developers. Pywavelets package, Dec 2017.