Predicting the stock market with Deep Learning

Valentin Courgeau 17098662 VALENTIN.COURGEAU.17@UCL.AC.UK
MRes - CDT in Financial Computing and Analytics

Giang Dang 17055511 G.DANG.17@UCL.AC.UK
MSc Data Science and Machine Learning

John Goodacre 13064947 JOHN.GOODACRE.13@UCL.AC.UK
MRes - CDT in Financial Computing and Analytics

Pier Francesco Procacci 17075285 PIER.PROCACCI.17@UCL.AC.UK
MRes - CDT in Financial Computing and Analytics

Abstract

We investigate the prediction of financial time-series using Deep Learning. In terms of motivation, much of the team had considerable background within the financial area and wished to apply the material from the course to address the very difficult problem of predicting noisy, financial time-series. To that end we referred to two papers. (Bao et al., 2017) and (Takeuchi & Yu-Ying, 2013) used different approaches but claimed excellent returns. This paper addresses our implementation of both papers and our findings.

The two implementations were:

- A classification approach utilising cross-sectional momentum as a feature applied to a ConvLSTM architecture. The aim being to identify stocks that will exhibit next week return above the average.
- A regression approach utilising a number of time varying features, this time in an attempt to directly predict tomorrow's index close. The architecture being a mix of Wavelets, Stacked Auto-encoders and stacked LSTMs.

We achieved positive results for both of our implementations. However given the difficult nature of financial time series advise caution in interpretation, we feel that the results whilst being promising, merit further investigation. In particular we felt the architecture in the second paper (Bao et al., 2017) may be overly complex and our preliminary findings appear to show that simpler models perform even better.

Group Project:
Introduction to Deep Learning, UCL 2017

1. Deep Learning for Equity prediction: A classification approach

In this section we refer to the paper Takeuchi & Yu-Ying (2013), where the authors combined the 'momentum effect' with the use of a deep architecture in order to predict 'bucket's of stocks. The momentum effect is a cross-sectional phenomenon and, by feeding the model with inputs consistent with this framework, we wish to predict next week performance of every stock with respect to the average of all the others in the index.

1.1. Background

1.1.1. The Momentum Effect

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 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 upon momentum is due to Levy (1967) who defined these strategies as Relative Strength strategies, before becoming popular even among academics during 90s (Jegadeeshn & Titman, 1993; Carhart, 1997).

Jegadeeshn & 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]

1.1.2. Long Short Term Memory - LSTM

In our implementation of both papers the final part of the model architecture was a stacked LSTM.

LSTM's were first introduced by Hochreiter, Sepp and Schmidhuber, (1997). 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 1 comes from the excellent blog by Colah http://colah.github.io/posts/2015-08-Understanding-LSTMs/, who describes the architecture in more detail.

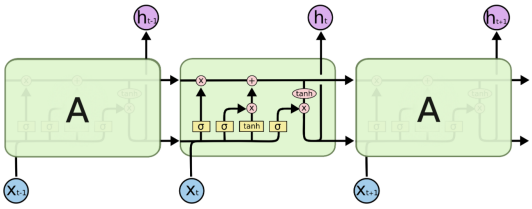

Figure 1. Illustration of a typical LSTM mechanism.

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.
- **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.

Convolutional LSTM is the combination of convolutional layers and LSTMs. The convolutional layers exploit spatial structure in the dataset and extract its hierarchical feature representations. In handling spatiotemporal domains, applying to LSTMs the feature representations extracted from convolutional layers instead of raw input often leads to better result. This architecture was implemented in the first part of the project.

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 on the second paper utilised stacked LSTM's.

1.2. Methodology

In Takeuchi & Yu-Ying (2013), the authors proposed a deep architecture consisting of a stack of Restricted Boltzmann Machines. While maintaining aligned purposes, inspired by recent publications (Hammerla et al., 2016; Yu-Guan & Ploetz, 2017) we deviated from Takeuchi & Yu-Ying (2013) by considering a different model architecture. Given the cross-sectional nature of momentum, we wish to exploit the properties of convolutional layers in considering local correlations and of LSTM layers in taking temporal context into account, while limiting hand-engineered features.

1.3. Implementation

1.3.1. Dataset specification

We restrict our analysis to the constituents of S&P 500 Index from January 1990 and December 2016.

[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

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.

1.3.2. 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-56. 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 3.

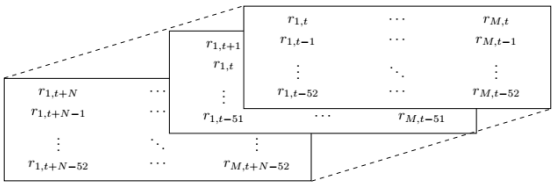

Figure 2. Dataset structure

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 3, 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.

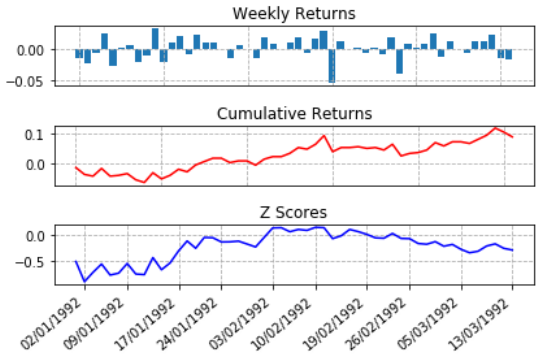

Figure 3. Preprocessing pipeline

1.3.3. Model

The final architecture we considered is constituted as follows:

- Input Layer
- Conv Layer: 16 feature maps, 3x3 kernel size, zero padding, with ReLU activation;

  Max Pool: 1x2

  Drop Out: 0.5 rate
- LSTM Layer: 32 units, with ReLU activation;

  Drop Out: 0.5 rate
- Output Layer: binary classification, with softmax activation;

It results in a binary classifier predicting, for every stock, next week's performance as defined by labels in previous section. We implemented the architecture in Keras.

1.4. Experiments performed

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.

In defining our final architecture design, we experimented with the choice of different hyper-parameters and regularization techniques gathering the following observations:

- Max-Pool and Drop-Out techniques enhanced significantly both accuracy metrics and financial performance;

- Augmenting the architecture dept or size by adding Conv or LSTM layers or by expanding layer dimensionality lead to worse performances;
- The *Adam* optimization routine (Kingma & Ba, 2014) provided the model with a stable and faster convergence;
- We fixed the number of epochs to 5. This decision comes by experience seeking balance to fit the dataset avoiding overfitting and limiting computational costs.

The overall intuition is that a lighter architecture performs better in both fitting the data and generalizing on the test set. As discussed in following sections, we believe that some these results are heavily affected by the size of our dataset.

1.5. Test Metric and Results

This section refers to the performance obtained in the test set in terms of both prediction accuracy and financial performance. We measured the F1-score for the binary classification of each stock and, as a measure of accuracy for the overall model, we considered the average across stocks

Also, we considered the realized returns on a simple long/short strategy based on the obtained predictions as a measure financial performance. Each week, 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 chose K usually from 1 to 4. In our experiment, instead, we considered 1 week holding period.

1.5.1. Prediction and probabilities

The model achieves an overall accuracy of 51.936%. The average weekly return provided by the long/short strategy is 0.11%. 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 4 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.

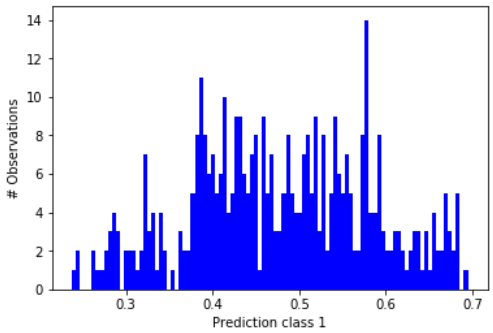

Figure 4. probabilities of being in class 1 for one stock in the test set

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 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 series of weekly returns of the strategy. Figure 5 presents the cumulative returns of the strategy and of the considered benchmark index (S&P500) from January 2008 to December 2016.

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.

[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.

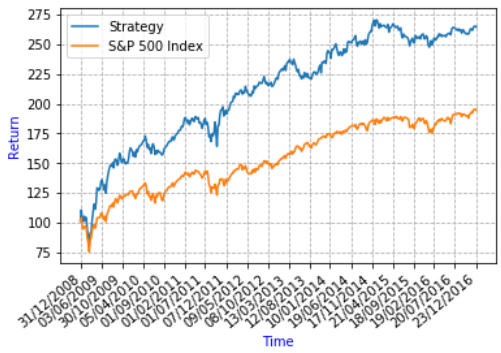

Figure 5. Cumulative returns indexed at 100 for proposed strategy vs benchmark

1.6. Conclusion

In this study we presented an application of ConvLSTM deep neural network to stock trading. The objective is to use a deep architecture to discover features in financial time series that can successfully predict future returns. While most research in deep learning focuses on tasks considered to be easy for humans, predicting stock returns is rather challenging given the low signal-to-noise nature of financial time series and given the efficiency of financial market to rapidly exploit existing patterns.

Our study illustrates that a ConvLSTM architecture can extract useful features from high noise-to-signal time series if the inputs are correctly preprocessed. On the chosen accuracy metrics we found positive results, similarly to (Takeuchi & Yu-Ying, 2013). The performance of the long/short trading strategy obtained using probabilities confirms that the model can be trusted in its ability to recognize signal over noise.

Future developments on this framework will be directed to examine the impact of updating weights as new data are available and to the use of a larger dataset. In the proposed experiment, the model is trained only once for each stock and the obtained weights are hold for the entire test period. Although being a standard approach, in this way we are not considering all the available information at each date and this is non-logical from an investor perspective since the market regimes change over time. In future experiments we will examine the effect of retraining the model at each rebalancing date while assessing the computational costs of this approach.

One major pitfall of our experiment is related to the small amount of data considered. Having restricted our investment universe to the constituents of the S&P 500 Index, we composed our dataset with those stocks that remained consistently in the index over the considered time interval (1990-2016). Since the index is periodically rebalanced, the longer the time interval considered, the lower the number of stocks that remained consistently in the index. This limited heavily our ability to consider a larger dataset.

In future work, considering the updating approach just discussed, we would like to consider a dynamic dataset taking into consideration new stocks entering the index on a yearly base. Every year, we can consider all of the constituents of the index and their related time series with all the available observations, train the model, construct the trading strategy accordingly and update the weights at each rebalancing date. Next year, we will consider the (possibly) new stocks constituting the rebalanced index with respective time series for the entire available time span. This framework would allow to consider more stocks and longer time series while maintaining an investment universe coherent with the index.

2. Deep Learning for index prediction: A regression approach

The second paper (Bao et al., 2017) approached index prediction from quite a different perspective. Rather than looking at cross-sectional momentum and using a neural network on a classification task. The authors used various data features as inputs to a fairly complex model. This time the output being a regression task, where the prediction was tomorrow's index close.

2.1. Background

Although performing regression in the stock market is nothing new, to our knowledge (Bao et al., 2017) were the first to implement the following particular architecture in this prediction task.

- **Discrete Wavelet Transform**: The intuition being to smooth or de-noise the input feature data.
- **Stacked Auto-Encoder**: The intuition being to take the smoothed feature data and reduce dimensionality, with stacking aiding the learning of more complex hierarchical structure.
- **Stacked LSTMs**: The intuition being to take

the smoothed, lower dimensional input features and learn temporal dependencies.

This model ends with a 10-d fully-connected layer with *ReLU* activation function to add extra non-linearity.

Following Bao et al. (2017) and our own experiments, adding any dropout did not seem a priority. We explain this by the general and non-flexible definitions of mother wavelet functions. This said, the impact of dropout could be interesting to quantify in the future.

2.2. Methodology

Our initial methodology was an implementation of the (Bao et al., 2017) paper. This we achieved and found similar positive results. We deviated from here by also examining whether this complicated architecture was genuinely helpful. There are a number of moving parts. Wavelets, Stacked Auto-Encoders, and Stacked LSTM's with a particular lookback period. We wished to test if each of these 'modules' was genuinely necessary and set about simplifying the model.

2.2.1. Cheat Feature

Financial time series are notoriously noisy. When one tests a network architecture on other types of data it is often easier to gain conclusive results and choose the right architecture. However, with financial time series we have a joint test of both the predictive ability of the initial data inputs, as well as the model architecture itself.

Thus, we introduced a new 'cheat feature' where future information was passed into the data inputs (effectively tomorrow's close multiplied by gaussian noise with mean one). This ensured that our input data did actually have predictive power embedded within it and helped us to examine whether the neural network was actually able to learn this useful feature. The secondary benefit enabling us to test whether the wavelets and Stacked Auto-Encoders (SAE) actually helped or obfuscated.

2.2.2. SAE 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 simpler 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.

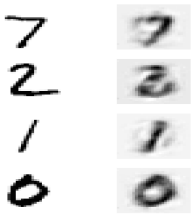

Figure 6. Quick sanity check, 5 layer stock market SAE applied to MNIST

2.3. Implementation

2.3.1. Dataset specification

The data used in the original paper (Bao et al., 2017) is the daily quotes of the S&P500 along with other technical indicators from 1st July 2008 until 30th September 2016 - 2079 datapoints. Due to length limit, we cannot add the features' description (see Bao et al. (2017), Table 1). We work with 19 predictors to predict tomorrow's close price.

2.3.2. Discrete Wavelet Transform

Idea Following (Bao et al., 2017), 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:

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

for $j = 1, \ldots, J$. For more detailed information see the original paper (Bao et al., 2017), pages 5-6.

Application Bao et al. (2017) only provided a few details regarding the implementation and application of DWT. As advised in Hsieh et al. (2011) and Bao et al. (2017), 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 time-series 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. (2011) 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 (Wasilewski & Developers, 2017) and the 1D Multilevel DWT functionality. This implementation relies on Mallat's algorithm (Resnikoff & Wells, 1998) for Fast Wavelet Transform computation.

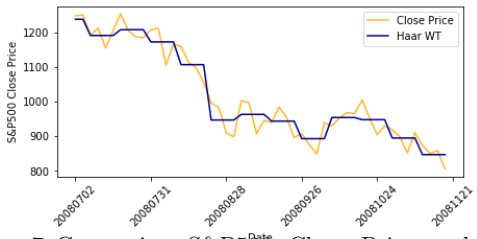

Figure 7. Comparing S&P500 Close Price and Haar wavelet transformed data on the first 100 dates

2.3.3. Stacked Auto-Encoders

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 (Bao et al., 2017), the authors 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 8 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.

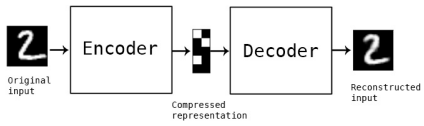

Figure 8. Simple Autoencoder

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. (2017) 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 the 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 9, from Bao et al. (2017) illustrates the exact architecture of the SAE module of our model.

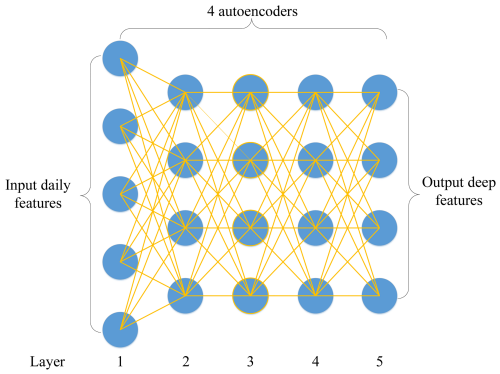

Figure 9. Stacked Autoencoder architecture as presented in Bao et al. (2017)

We implemented the SAE in tensorflow, closely following the implementation of Bao et al. (2017), including network structure, dimensionality and util-

ising regularisation. We however found it absolutely necessary to normalise the data, and found that we achieved more stable results using a tanh activation function rather than their use of the sigmoid.

2.3.4. Stacked LSTMs

We described stacked LSTMs in part 1 of this paper. In this particular implementation the outputs from the SAE's were inputs into four stacked LSTMs, each comprising ten dimensions. The output from this being our final regression predicting tomorrow's index close.

2.3.5. Implementation Pitfalls

With hindsight the implementation sounds rather smooth, however this covers various sensitivities we found with this architecture and particular data set.

- We found data normalisation to be essential.
- LSTMs took significantly longer to train, but were far more sensitive as regards stability.
- We required batch normalisation for the stacked LSTMs. This counteracts the strong non-stationarity of the financial timeseries.
- We found the tanh activation gave more consistent results for the SAE's than Bao et al. (2017) use of the sigmoid. tanh is less polarised than sigmoid, leading to smoother data.
- We coded a baseline multi-layer perceptron - this was not only quicker to train and test but surprisingly appeared to give more robust results.
- With such a complex model (3 models in one, all of them stacked in various ways), we found sensitivities to hyper-parameters and architecture that would have required a great deal of time to explore fully.
- We sped up a great deal when we were able to watch the losses on the training and validation set, particularly when learning became unstable.
- We used the *Adam* optimisation routine (Kingma & Ba, 2014) - with a learning rate of 0.002. The learning rate found by experience seeking optimisation stability, the optimiser chosen because it is known to work well with LSTMs using the Mean Squared Error (MSE) loss function.
- Due to the short sequences involved we found it useful to unroll sequences when training the LSTMs.

2.3.6. Architecture

The final architecture was thus input feature data, to wavelet transformations, to SAE's, to stacked LSTM's resulting in a regression predicting tomorrow's close of the S&P500:

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;

2.3.7. Training specifics

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 was fixed to 1200 by trial and error using our 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.

2.4. Experiments performed

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 trained the stack of LSTM's and predicted the next day's index close. We would like to find the lookback parameter which provides the best predictive power using metrics (see next subsection) and the buy-and-sell strategy (Appendix A).

Impact of the modules Wavelet transforms (Ramsey, 1999) and Stacked Autoencoders (Chen et al., 2014) are powerful denoising modules used respectively in timeseries analysis and computer vision. We wanted to replicate a model comparison of the WT-SAE-LSTM model with sub-models constructed with/without certain modules on the *test*

set. We also included *cheat models* as in the lookback parameter discussion above and an Multi-Layer Perceptron as a baseline model.

2.5. Test metrics and results

We compare models using 3 standard metrics: MAPE, R and Theil U. For true index close $(y_i)_{i=1}^N$ and predicted index close $(y_i^*)_{i=1}^N$:

$$MAPE(y, y^*) = \frac{1}{N}\sum_{n=1}^{N}\left|\frac{y_n^* - y_n}{y_n}\right|$$

$$R(y, y^*) = \frac{\sum_{n=1}^{N}(y_n - \bar{y})(y_n^* - \bar{y^*})}{\sum_{n=1}^{N}(y_n - \bar{y})^2(y_n^* - \bar{y^*})^2}$$

$$TheilU(y, y^*) = \frac{\frac{1}{N}\sum_{n=1}^{N}(y_n - y_n^*)^2}{\sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n)^2} + \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n^*)^2}}$$

MAPE measures the size of the error, R the correlation between y and y^* and Theil U is a *relative measure of the difference between two variables*. We would like R to be large and R/Theil U to be small. Also, we used a simple buy-and-sell trading strategy as explained in A 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. 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 For this test, we obtained the buy-and-sell cumulative returns pictured as follows for lookbacks of 2, 3, 4 and 5: We observe a

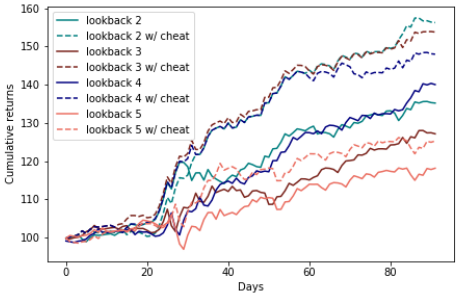

Figure 10. Cumulative returns indexed at 100 on the validation set for lookback parameters 2, 3, 4, 5. Dashed lines for cheat models, solid lines for original models.

similar performance of cheat models (except with a lookback of 5 which may be too complex for our small dataset). They also provide the best 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 From Figure 11 and Table 1, we observe that the more complex the model the worse it performs using the cheat feature or not. Indeed, the MLP baseline perform best with stellar test metrics, returns and training time. The crude stacked LSTM with a lookback of 4 is also performing well return-wise although the LSTM with a lookback of 2 has better test metrics. This is probably due to the higher complexity of the former, allowing it to guess the price move direction better yet giving less precise prediction (due to increased training difficulty from limited data). The largest model yields small returns although performing better than just SAE-LSTM, indicating that WT does provide some denoising. Every time we include the SAE module, the model does not perform well. We also note that all cheat models performs significantly better except for WT-SAE-LSTM for which the opposite happens. This could indicate that the cheat feature transformed through WT-SAE misled the model. This showcases one of the flaws of a complex model.

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*

2.6. Conclusion

We first implemented the paper by Bao et al. (2017). On their chosen metrics, MAPE, R, Theil U and a long/ short trading strategy with transaction costs, we found that for the test period in question results were positive. In financial terms sometimes exceptionally so.

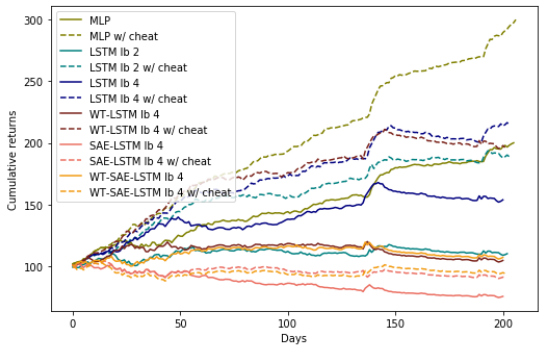

Figure 11. Cumulative returns indexed at 100 on the test set for lookback parameters 2 and 4 and with/without wavelet tranform (WT) and Stacked Autoencoders (SAE). Dashed lines for cheat models, solid lines for original models.

However, we felt during implementation that the model was complex - it was difficult to ascertain which part of the overall model - 'input data features', 'wavelet denoising', 'stacked auto-encoder' or 'stacked LSTM' was contributing what. So we implemented a further set of tests.

The first set of tests being to strip away the wavelet denoising and SAE dimensionality reduction modules and examine the impact of the lookback period on the LSTMs performance. By its nature the LSTM is seeking temporal dependencies and as shown in our results the length of the lookback window gave considerably varied results.

The second set of tests was a further examination of the merits of this architecture and examining various combinations of the modules. This involved using a *cheat* feature (Section 2.2.1): as intuition would suspect, this magnified results. The conclusion of our second tests seemed somewhat more negative, at least as regards the paper introduced by Bao et al. (2017). Indeed our preliminary results showed that the simpler the model the better the performance. The five layer stacked autoencoder simply appeared to 'mash' features in this dataset rather than add real value. The Wavelet denoising did less damage, but somewhat ironically we found that simpler models where normalised input data went straight into the LSTM's or MLP's without the 'advantage' of smoothing or dimensionality reduction appeared to perform better in this specific context.

The final point is a philosophical one. Blindly taking financial input data and pushing it through a neural network (or stack of them) in order to predict tomorrow's closing price makes a considerable set of implicit assumptions. We are first assuming that the input data contains information that is predictive of the output and second that there actually exists some complex non-linear function which provides an approximate mapping to tomorrow's closing price. Let alone the implicit assumptions about the well behaved statistical nature of the data. We were actually pleasantly surprised to see some of the positive results, but still believe questions remain over the merits of this particular architecture.

3. Overall Conclusion and further work

We were interested in handling financial timeseries from both classification and regression perspectives by implementing predictive models, respectively inspired from Takeuchi & Yu-Ying (2013) and Bao et al. (2017) based on the LSTM module.

For the latter, we provided a careful study of different submodels using prediction metrics and a buy-and-sell strategy. It allowed us to highlight the importance of simple models with restricted dataset and limits of a double denoising procedure. One potential approach could be to add convolutional neural networks to access a higher abstraction of features (Chen et al., 2016). To our knowledge, the potential symbiosis with wavelets and autoencoders is yet to be determined. Finally, the LSTM is currently the state-of-the-art for timeseries yet Deep Residual Learning could provide improvements (Borovykh et al., 2017).

For the former, we constructed an ad hoc dataset and experimented with a ConvLSTM architecture. To our knowledge, this is the first time this model is used in predicting financial time series. Coherently with the previous, we found that restricted datasets typically considered in financial research and industry are exposed to overfitting when fed into a deep architecture and, thus better served by lighter architectures. Our study on prediction probabilities suggests that results of classification tasks should be carefully interpreted when related to highly noisy data.

Overall, we concluded that deep learning architectures can represent a valuable framework in dealing with financial time series in both regression and classification tasks. In particular, we illustrated the potential for deep learning to extract discriminative information from data without the labor-intensive feature engineering activity that has characterized the field over the past decades.

A. Buy-and-Sell trading strategy

Denote y and y^* respectively the true and predicted close price timeseries. Given in Bao et al. (2017), 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

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

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

Borovykh, Anastasia, Bohte, Sander, and Oosterlee, Cornelis W. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

Carhart, M. On persistence in mutual fund performance. *Journal of Finance*, 52(1):57–82, 1997.

Chen, Jou-Fan, Chen, Wei-Lun, Huang, Chun-Ping, Huang, Szu-Hao, and Chen, An-Pin. Financial time-series data analysis using deep convolutional neural networks. In *Cloud Computing and Big Data (CCBD), 2016 7th International Conference on*, pp. 87–92. IEEE, 2016.

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

Hammerla, Nils Y., Halloran, S., and Ploetz, T. Deep, convolutional, and recurrent models for human activity recognition using wearables. *CoRR*, abs/1604.08880, 2016. URL http://arxiv.org/abs/1604.08880.

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

Hsieh, Tsung-Jung, Hsiao, Hsiao-Fen, and Yeh, Wei-Chang. 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.

Jegadeeshn, Narasimahan and Titman, Sheridan. Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, 48(1):65–91, 1993.

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

Levy, Robert A. Relative strenght as a criterion for investment selection. *Journal of Finance*, 22(1): 595–610, 1967.

Ramsey, James B. 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.

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

Takeuchi and Yu-Ying. Applying deep learning to enhance momentum trading strategies in stocks. 2013.

Wasilewski, Filip and Developers, PyWavelets. Pywavelets package, Dec 2017. URL https://github.com/PyWavelets/pywt.

Yu-Guan and Ploetz, P. Ensembles of deep LSTM learners for activity recognition using wearables. *CoRR*, abs/1703.09370, 2017. URL http://arxiv.org/abs/1703.09370.