Predicting the stock market with Deep Learning: Individual addendum

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

Abstract

This paper presumes that the reader is familiar with the content of the group paper 'Predicting the stock market with Deep Learning'. The purpose of the paper is to expand upon the original paper. This expansion will serve three aims. First, a more detailed story of the implementation. Second, a critique of the approach. Third, further expansion on the interpretation of results. Given that we are asked to illustrate that we fully engaged with the project on an individual basis, the author will also offer an honest view of where strong contributions were made to the overall group project.

1. Project selection

Motivated by the course but aware of the particular difficulties involved with financial time series, we decided to use Deep Learning to explore this arena. We quickly found two interesting papers (Bao et al., 2017) and (Takeuchi & Yu-Ying, 2013). Although it possibly doubled the workload we decided out of interest to implement both papers. The detailed description of both papers is given in the group project.

- (Takeuchi & Yu-Ying, 2013) took a known feature in the stock market, 'Momentum'. The authors then constructed a model comprising a restricted Boltzman machine and an LSTM. The key to their methodology was a cross sectional approach and also classification based approach where stocks were placed in buckets expected to out or under-perform the index.
- (Bao et al., 2017) used a regression based approach. Input data comprising around twenty common financial market features. Their model was a small 'production line', comprising Wavelet transformations, Stacked Auto-Encoders, and Stacked LSTM's. The goal being a prediction of tomorrow's index close.

Because we were implementing two separate (albeit related) papers, we chose to operationally split into two teams of two for primary implementation responsibility, coming together where we lacked clarity. Myself and Valentin implemented the (Bao et al., 2017) paper and Pier and Giang the (Takeuchi & Yu-Ying, 2013) paper. The idea being to implement and if time served to cross-fertilise at the end, dependent upon our findings.

2. Implementation

2.1. Deep Learning for index prediction: A regression approach

2.1.1. Wavelets

The implementation of (Bao et al., 2017) was done by myself and Valentin. In terms of the initial implementation Valentin concentrated on implementing the wavelet transforms. This was relatively straight forward and involved simply importing the 'pywt' python wavelet package, having the data in the correct format and applying a 'Haar' wavelet twice. Slightly irritatingly, only broad implementation details were given by (Bao et al., 2017).

2.1.2. Stacked Auto-Encoders

The SAE's were implemented by myself in Tensorflow. (Bao et al., 2017), used a five layer SAE. As I mentioned in the group report when we stack auto-encoders we are effectively nesting them. I

trained the outer layer first and instead of a hidden latent layer, nested another auto-encoder each time I stacked. The middle latent layer would be our final output. The inputs of course being the outputs from the wavelet module. Because financial data is so noisy and because I needed to test the implementation was actually working. I applied the same architecture to the MNIST dataset, checked the mean squared error and added the decoder back to 'eyeball' the reconstruction of test images.

SAE implementation details: I had a few design decisions and stumbling blocks along the way. In addition, I didn't start with a totally blank screen with the code (I am not yet a tensorflow expert), but worked off simple examples on the tensorflow website and elsewhere, then built-up from there.

- **Optimiser**: No issue here, Adam - learning rate between 0.005, 0.002. The loss function to optimise being based on Bao et al. (2017), a mean squared error with L2 regularisation.
- **Data**: I found it necessary to normalise the data, otherwise I had unstable results.
- **Activation function**: (Bao et al., 2017), used a sigmoid activation. However, for me this sometimes gave unstable results. When I changed to tanh, there were no more issues.
- **Generalisation**: I was keen to be able to play with different architectures, and dimensions. (Bao et al., 2017) used four SAE's each of ten dimensions with inputs of around twenty dimensions (dataset dependent). I ensured the architecture was not 'hard-coded' but could be passed into the SAE construction like any other parameter.

As mentioned in the group write-up, the intuition behind auto-encoders is to find a 'good' latent representation of input data, by stacking them we also hope to capture hierarchical information.

2.1.3. Stacked LSTMs

The stacked LSTM's were also implemented by myself. This time I made life far easier by using Keras. Again I used the Adam optimiser and mean squared error for the loss function. (Bao et al., 2017), used four stacked LSTMs. Dimensions were again low with ten dimensions. I also implemented a Multilayer Perceptron and again passed in the network architecture as parameters to enable generalisation and easier testing.

LSTM implementation details: Keras is much easier to work with than Tensorflow in my opinion. The main parts that slowed me down were to do with data and a few settings.

- **Lookback**: We are passing normalised daily data into stacked LSTM's in a sequential fashion. For the LSTM's there was effectively an extra dimension called a lookback (where we would lookback say four to five days). This is relatively easily handled in Keras by setting up the data correctly and adding a dimension.
- **Initially bad results**: The LSTM output was our final prediction. As described in the group report, we used a training set and validation set, with Valentin helpfully coding a live graph so we could observe if the LSTM was actually learning. Initially the output was rubbish (often a flat line prediction). I solved this by doing the following:
  - **Batch Normalisation**
  - **Unrolling sequences** For some reason this stabilised results, I am still unsure why).
  - **Added Dropout** This was not necessary for stability, just aiding generalisation.
- **Simpler Models** In attempting to solve various LSTM issues, I felt the need to introduce a simple baseline model. I implemented a simple multi-layer perceptron and dropped all the Wavelet and SAE stuff. Although this gave positive results and the modular approach enabled me to see that each part was implemented correctly, I still didn't find the results completely compelling. Thus, to further examine whether the data was not the best predictor, or whether the model simply wasn't learning good predictive features, I invented and implemented a 'cheat feature'.
- **Cheat Feature** In finance, it is anathema to let future information 'leak' into your model. In this case I deliberately chose to do so with interesting results. Given that we are trying to predict tomorrow's close with complex models and input features, the 'cheat' was to add tomorrow's close to the input data (with varying degrees of gaussian noise). The reasoning being that if the 'answer' was mixed in with the question and the model could not 'learn' then we

definitely had some issues! Happily this worked well, and by varying the degree of gaussian noise we were able to examine whether the model lost predictive power. It also served to 'prove' our implementation correct. In the group write-up we show performance both with and without the 'cheat feature'.

2.1.4. Test Metric Details

Valentin coded up the test metrics from (Bao et al., 2017), as well as the trading methodology (including transaction costs). I coded up a method to extract each days trading performance so we could examine the overall performance through time. I was very keen to make the system end-to-end, which I did, whilst Valentin concentrated on further generalising the code so that we could pass test cases into the whole end to end structure using a yaml file.

This gave us an end-to-end system. Which we could test by passing in a yaml file of hyper-parameters and architectures, and automatically save our results and models.

2.2. Deep Learning for index prediction: A classification approach

The implementation of (Takeuchi & Yu-Ying, 2013) was really done by the other half of our team, Pier and Giang. As a quick reminder from the group write-up, the implementation was a classification based approach to 'bucket' stocks on a cross-sectional basis.

(Takeuchi & Yu-Ying, 2013) implemented a model comprising restricted Boltzman machines and LSTM's. After early consultation with the whole team, Pier and Giang wished to implement the cross-sectional momentum feature using Convolutional LSTM's. The original paper is now four years old, and we all agreed it would be interesting to test a more modern architecture.

2.2.1. Convolutional LSTM's

Convolutional LSTM's are described in the group write-up, but they are a reasonably simple variant on vanilla LSTM's. We simply have convolutional layers. If one imagines a plain vanilla LSTM, then it can be less easy to model spacial structure such as images. Typical uses of a Conv-LSTM might be areas such as activity recognition in video, or image and video description. They have also been used in areas such as natural language processing, or audio.

Given Pier and Giang did the vast majority of the implementation of the other paper. I think it would be slightly dishonest of me to comment much further on this part of the implementation.

3. Result interpretation

3.1. Classification Approach

I believe the key interpretation of the results by the team was largely correct. That is for our implementation of (Takeuchi & Yu-Ying, 2013), that the overall performance over the 8 year period massively outperformed the index. The cross-sectional momentum feature was useful, the Conv-LSTM model picked this up and exploited it out of sample. I do however have further comments in my critique.

3.2. Regression Approach

For our implementation of (Bao et al., 2017), the results were also positive. However, by going further than (Bao et al., 2017) and varying the architecture itself - both as regards the look back period, and as regards the impact of each different module (Wavelet, SAE, LSTM), we raised a number of questions. These appear to weaken the conclusions of the paper (Bao et al., 2017), suggesting that simpler models perform even more strongly for this particular dataset. I consider these questions raised, but not properly answered.

4. Critique

4.1. Approach

By implementing two papers, we doubled our workload and although there was continuous communication, we became slightly silo'ed. Although we all 'understand' the details, I personally believe you only have genuine understanding of all the pitfalls of the parts you yourself implement.

Given more time one extension would have been to look at ways of ensembling the momentum Conv-LSTM results with the Wavelet-SAE-SLSTM prediction results, (for example much of the same data features from the (Bao et al., 2017) paper are also

available for individual stocks).

4.2. Paper Selection

I realised half way through how much easier life would have been if we had chosen very strong papers to implement. The full implementation details of (Bao et al., 2017) were not elucidated. Indeed we found mistakes in the paper such as their being paid for transaction costs. We ended up with a number of implementation questions.

With the (Takeuchi & Yu-Ying, 2013) paper, we felt that the restricted boltzman machine - LSTM approach was less interesting than the Conv-LSTM approach, and certainly less modern. In this case it is not that the paper is bad, just not the most modern.

4.3. Implicit Logic

Having a background in finance, I worry slightly about the implicit logic behind for example the (Bao et al., 2017) paper. Here we have a very noisy time series dataset. The features are low dimensional, only around twenty. The model involved two Haar wavelet transformations, a five layer Stacked Auto-encoder, and four stacked LSTMs. Given such low dimensional features and a look back period of no more than five days, I personally question the logic of the approach. I understand the intent behind each module, but felt it over-complicated. Indeed one of our key conclusions is raising the question as to whether simpler models would do better, seemingly so if our initial findings hold true.

The final aspect of these papers is that when we are looking at financial time series data, they can be modelled as stochastic processes. Often very noisy stochastic processes. As I mentioned in our conclusion, (Bao et al., 2017) are making some strong implicit assumptions about the input data containing predictive features, and about there actually existing some non-linear function map between the input data and an approximation of tomorrow's closing price. Finally of course the statistics of financial data can be ill-behaved. I feel these papers, ignore these implicit issues and that they could be genuine issues.

5. Unanswered Questions

5.1. To what extent are these results robust?

Is it the case that the positive results were simply that a particular feature happened to work in that time period (for example momentum) and that in other periods results would be less strong or even negative?

If one looks more closely at (Takeuchi & Yu-Ying, 2013) and our team's implementation. The team made a positive choice in tailoring the classification of stocks to where the model had the greatest confidence. However, I question the robustness of results over the whole time period in question. The starting period for test purposes was late 2008, the bottom of the market occurred in early 2009, followed by a quick recovery. This was an unusual period. I would ask whether a significant portion of the positive results were gained due to this environment and would wish to test the model in different time periods.

I would also explore the logic behind having an 8 year training period and a subsequent 8 year test period. The implicit assumption here is that markets do not exhibit 'regime shifts', at least as regards the momentum effect and that it is a positive factor throughout the 16 year period. My own preference would be to have a 'rolling window' approach so the model has always been trained on the most up to data information.

5.2. What is the right architecture for this type of problem?

Neural nets can over-fit or 'learn' the training data and not generalise (hence regularisation and dropout). But financial data is not like visual data - why should a Conv-LSTM a good model? Is it particularly good at picking up a signal/ noise? How should we choose the dimensionality of these models relative to the size of the data? Why did the Stacked Auto-encoders appear to make results worse and just 'mash' features? In machine vision there is a clean logic behind CNN's and say translational equivariance in images. Finance is much more opaque.

In summary, although we had excellent test results with novel approaches added to both papers. Our work also raises further questions, rather than giving some definitive answer.

References

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.

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