Financial Time Series Forecasting – A Deep Learning Approach

Alexiei Dingli and Karl Sant Fournier

***Abstract*—This paper is intended as a follow up to a previous study of ours - Financial Time Series Forecasting - A Machine Learning Approach. The aforementioned study evaluates traditional machine learning techniques for the task of financial time series forecasting. In this paper, we attempt to make use of the same base dataset, with the difference of making use of a novel branch of machine learning techniques known as Deep Learning. These techniques have been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. These deep architectures are known to excel in tasks such as image and text recognition, but have not been exploited as much in the field of finance. In particular, for this study we will be making use of Convolutional Neural Networks (CNNs) to forecast the next period price direction with respect to the current price. We achieve an accuracy of 65% when forecasting the next month price direction and 60% for the next week price direction forecast. Whilst these results are anything but random, we are not able to match or surpass results obtained by industry leading techniques such as Logistic Regression and Support Vector Machines.**

***Index Terms*—Data science, deep learning, fintech, machine learning, stock market.**

I. Introduction

The Stock Market is known for its volatile and unstable nature. A particular stock could be thriving on one day and struggling on another. Big money is made from selling stocks when they are at their highest and buying when they are at their lowest. The logical question would be: "What Causes Stock Prices To Change?". At the most fundamental level, the answer to this would be the demand and supply for a certain stock affects the stock price. In reality, there are many theories as to why stock prices change, but there is no single theory that explains all, simply because not all stocks are identical, and one theory that may apply for today, may not necessarily apply for tomorrow.

Since the introduction of ANNs, other techniques such as Support Vector Machines have been given more importance importance and have achieved similar and better results in a shorter timeframe. However, Deep learning has been getting a considerable amount of attention in recent years. Namely Google, Microsoft and Facebook are investing heavily in this area, through a number of open source projects. Modelling complex problems using ANNs necessitates the use of many layers due to the nature of the problem at hand. The introduction of deep learning techniques facilitates training of these deep networks through a number of approaches. There exist a number of deep architectures such as Deep Belief Networks, Recurrent Neural Networks and Convolutional Neural networks amongst others. For the purpose of this study, we will be exploiting CNNs for classifying the next period price direction.

Manuscript received July 24 2017; revised November 3, 2017.

The authors are with the Department of Artificial Intelligence, University of Malta, Malta (e-mail: alexiei.dingli@um.edu.mt, karl.sant-fournier.10@um.edu.mt).

II. Literature Review

In the field of deep learning, CNNs are inspired by the visual cortex and are one of the most important deep learning models. Since CNNs are based on the architecture of the visual cortex, they are highly effective in tasks such speech and Image classification [1]. Yan LeCunn is said to be one of the fathers of deep learning, mostly due to his work with a pioneering 7-Level CNN known as LeNet-5 in the year 1998 [2]. His work was applied by various banks in order to recognise hand written digits on cheques. In a CNN, neurons are arranged in such a way to have overlapping regions [3] which allow for a local relationship between their adjacent/neighbour nodes. The network consists of a convolutional, pooling, ReLU, fully connected and loss layers (See Fig. 1). Since the training of CNNs is computationally expensive, a number of researchers have exploited the use of GPU technology for parallel computation.

Moving over to the FinTech industry, [4] uses CNNs to make predictions for stock price changes based on the image of the time series plot. The author also attempts to colour code the time series, however the results of this approach were not positive. On the other hand, [5] use CNNs in a bank telemarketing case study, whereby the aim is to predict whether a customer will take up a particular marketing campaign based on a number of numeric and nominal features per customer. The results for this study yield an impressive 76.70% accuracy, which yields the highest accuracy amongst 7 classifiers. In order to incorporate external features in the forecasting model, [6] use a deep convolutional neural network to model short and long term influences of events of stock price movements. Results from this study show that CNNs can capture longer-term influence of news events than standard feed-forward networks.

Many researchers have taken an SVM approach to Financial time series forecasting. Whilst still using the CNN layered architecture, [7] demonstrates the benefit of replacing the softmax activation function with a linear support vector machine. The learning minimizes a margin-based loss instead of the cross-entropy loss. The author manages to achieve small but consistent improvements in typical deep learning problems such as MNIST and CIFAR-10.

doi: 10.18178/ijmlc.2017.7.5.632

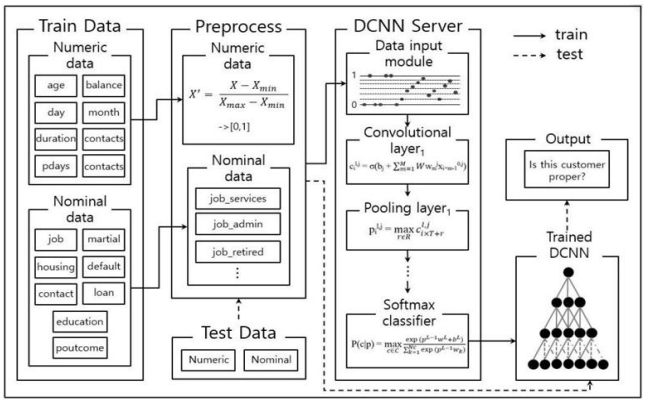

Fig. 1. System setup for bank telemarketing marketing study [5].

III. Methodology

A. Feature Extraction

As in any machine learning problem, once basic knowledge in the subject area is acquired, the next step is identifying the features to be used for our predictive model. At first glance, one may note that the price of a stock is affected by a number of phenomena. This section delves into the various data sources used for experimentation, including information relating to the stock itself along with other external data.

1) Historic prices & technical indicators

Historic stock prices are a good place to start, given that the data is publicly available through a number of sources. One of the most widely used mediums is yahoo finance, which provides unlimited historic daily prices in csv format on demand. A python API known as yahoo-finance is used to retrieve these historic prices from 2003 till 2016, which are then stored in our relational database for future use. These historic prices are then used to formulate what are known as technical indicators. These indicators consist of various financial equations using historic prices which are used to paint a clearer picture as to the condition of the company for that point in time. Investopedia describe these indicators as any class of metrics whose value is derived from generic price activity. Technical indicators evaluate price levels, direction and momentum amongst others. Rather than developing custom functions for each indicator, a library called TA-Lib was identified. This library openly provides 200 technical indicators, and is designed in the form of an open source API. Given that python is the language of choice for experimentation, a Python wrapper for this API is used. The technical indicators used in our approach include a combination of momentum, volume, volatility and cycle based indicators. As their name suggests, each class of indicator focuses on a different aspect of market activity.

2) Currency exchanges

When investors purchase foreign equity, they are actually placing a bet on two elements: one on the stock itself and the other on the currency the stock trades in [8]. In order to evaluate this hypothesis, we collect a number of the most popular exchange rates to incorporate within our predictive model. Currencies were collected from yahoo finance and Quandl data repository, and these include: EUR/USD, GBP/USD and BITCOIN/USD. As one may note, USD was taken as the benchmark since it is one of the most prominent currencies in the global stock exchange.

3) World indices

Investopedia describes an index as a measure of change in a securities market [9]. Indices consist of a hypothetical portfolio containing a group of securities that generally represent the performance of the overall market. Typical examples of such indices include the Standard & Poor's 500 (S&P500), Dow Jones Industrial Average (DJIA) and the Nasdaq Composite Index (NASDAQ-100).

4) Commodities

A number of popular commodities were selected as features for our experiments. Prices for Gold and Oil commodities are retrieved from the Quandl data repository, store them in our relational database and incorporate them into our dataset as time series data.

B. Feature Selection

Now that various data sources and features have been identified, the next step is to integrate them with our dataset and evaluate which are most important for the problem at hand. In total, we have over 70 features comprising of technical indicators (Overlap, Volume, Momentum, Volatility and cycle), currency exchanges, commodity prices and world in- dices. Rather than passing all features to our predictive model we use feature selection techniques in order to statistically identify the most relevant features for the data at hand.

C. Dataset Balancing, Training, and Test Split

We will be utilising data from 2003 till 2013 (11 years) as our training set and 2014 till 2016 (3 years) as our test set. This ensures a substantial amount of data for training, whereby 11 years of stock movements should cover a wide range of long and short term trends. The remaining three years will be used for testing, this approach will ensure that we are forecasting and calculating our evaluation metrics on unseen, out-of-sample data.

D. Forecasting Model

After having tested a number of state of the art machine learning algorithms in our previous paper, a deep learning approach will be taken to solve the problem of financial time series forecasting. Various techniques will be tested out, involving different combinations of the features mentioned in Section III.A. We will be making use of a convolutional neural network as our deep learning classifier. Given the state of the art results being achieved through these techniques we expect to achieve similar or better results than those obtained using traditional machine learning methods.

Our CNN model is designed and developed using the TensorFlow. This open source library facilitates the training of deep neural network infrastructures such as CNNs. This CNN framework provided by TensorFlow is is primarily based on softmax regression. Softmax is also known as multinomial logistic regression. In standard logistic regression, classification is made using 1 or 0. On the other hand, the softmax function outputs probabilities of the example belonging to each class.

By means of this network, we aim to achieve similar, if not

better results than those achieved by algorithms such as Random forest and Support vector machines. We firstly discuss the various transformations done to the data for it to be in a format accepted by the network, we highlight the main elements of the network setup and lastly we explore the various permutations and forms the network can take, in order to fine tune and increase accuracy

1) Data preparation for CNN

Firstly, the base dataset that we make use of here, is identical to that was utilised for previous classification experiments, however with some minor transformations. Given that CNNs are designed primarily for images, we transform our dataset into a square-like shape of 8x8, representing in 64 features. With regards to our output, TensorFlow works comfortably using one-hot vectors. This representation involves having an multi dimensional array consisting of 1s or 0s, the index on which the 1 is found represents a particular outcome. The below example illustrates how our output is formulated:

- Price Movement Direction 'Up': [1,0]
- Price Movement Direction 'Down': [0,1]

2) Weights initialisation

Weights were initialised to a small positive value since we are using the ReLU (rectified linear unit) function as initialising this to 0, may result in so called 'dead neurons'.

3) First convolutional layer

We will be computing 32 features for each 2×2 patch with a stride of 1 so that we output a total of $8\times8\times32$ features. In order to ensure that we do not drop out any features during convolution, we apply 'SAME' padding. A bias variable has also been included for each output channel. To simplify the information from the convolutional layer, we then apply max pooling to the first convolution, reducing the shape down to $4\times4\times32$.

4) Second convolutional layer

The second convolutional layer will work in a similar manner to the first, however this time, rather than having 1 input, this layer will now have 32 inputs in order to cater for the output of the first layer. From these inputs we apply convolution and compute 64 features for each 2×2 patch. Padding is applied for this layer too, so as to not drop any features. The shape after applying the convolution is now have $4\times4\times64$ features. We then apply max pooling, and reduce it down to $2\times2\times64$.

5) Fully Connected Layer

Now that, through the two convolutional and pooling layers, the image size has been reduced to 2×2, we add a fully connected layer to our network. This layer connects every single neuron from the max-pooled layer to every one of the possible output neurons (the ranking threshold, One-hot vector bits). Fig. 2 illustrates the network we have just described in graphical format.

6) Training

Our training data is that of approximately 80,000 per industry for the daily dataset, due to the size of the data, training will be done in batches of 50. The number of steps required for training is calculated by dividing the length of the dataset by the batch size (in our case, 50). During the training process, the program will display the training ac- curacy and loss on the training data with every batch. This will serve us as an indicator, as to whether the networks accuracy is increasing/decreasing with the number of training examples fed into it.

7) Calculating accuracy

Once the model is finished training on the examples, the accuracy on the test data is computed by checking if the argmax of the predictions and the testing rankings are equal. Argmax is a function provided by the TensorFlow library, which returns the index of the highest entry in a tensor. In our case it is designed to return the index of the one-hot vector (where the '1' lies in the array). If the argmax for both the predictions and correct outputs equal, the 'equal' function will return a '1', meaning the model correctly predicted the ranking threshold for the image. This will result in an array of 1s and 0s as booleans, each value in the array represents a test example, indicating whether the prediction is correct ('1' if value is correct) or not ('0' if value is not correct). From this array, we can then retrieve the accuracy by calculating the mean. (If the array is [0, 0, 1, 0], the resulting accuracy is 0.25).

8) Fine tuning the network

Once the base network described above is functioning, and we hope to achieve results which are better than random chance, the next step is to fine tune and optimise the model in order to better the accuracy and efficiency. A number of approaches are taken in order to accomplish such a task. These include:

- Adjusting the depth of the network
- Adjusting the Learning rate
- Adjusting the size of the local receptive area
- Adjusting the number of features for each convolution

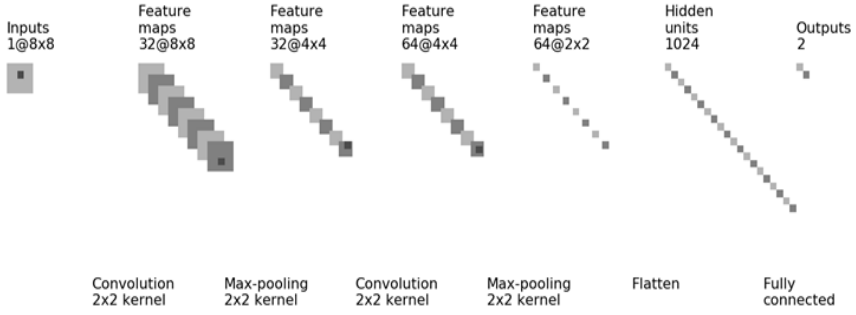

Fig. 1. Structure of Approach taken for CNN.

IV. Results And Evaluation

As described in the methodology, for our CNN implementation, we utilise the same features as before but structure them in such a way to form an 8x8 matrix to be fed into the network. After running a number of tests using the network setup described, it was noted that accuracy fluctuates from 54% to 64% for the monthly dataset. Training and test sets remained identical throughout these tests, after investigation it was noted that these fluctuations originate from the random initialisation of the weights for each convolutional layer. Due to this, we will be performing 10 runs for any alterations made to the network structure, and we will take an average accuracy over these 10 runs for comparison. In order to optimise the network, we alter the network parameters by means of adding or removing layers, adjusting the receptive fields for our convolutions, modifying the number of features that are extracted from each convolution and by adjusting the learning rate. We start by utilising the CNN approach for the monthly dataset, this was chosen given that it is the most 'predictable' classifier with a substantial number of examples. We are aware that the architecture of CNNs and any deep learning algorithm thrives when utilised on a substantially sized dataset.

A. Monthly Dataset

We start out by testing the Monthly Tech Dataset, with the model as described in the methodology, whereby we achieve an overall accuracy of 56%. This result does not match or surpass the 69% obtained by the MLP classifier. We assume that this discrepancy is originating from the more complex architecture of the CNN. In order to attempt to improve on the 56% accuracy obtained by the original CNN model, we conduct a number of tests, each time altering the various parameters and settings of the CNN.

We attempt to reduce the complexity of the CNN by reducing the number of layers from two to a single convolutional layer, through which we see accuracy increase substantially to 62%. This increase in accuracy due to a decrease in complexity substantiates our initial theory whereby we assume that the Multi Layer Perceptron achieves better results through its simpler structure. The second element tweaked was that of the learning rate, whereby the initial model had a learning rate of 0.001. We attempt to higher and lower the learning rate to 0.1 and 0.001, achieving 61% and 53% respectively, both settings not achieving better results than the initial 0.001 learning rate parameter. The adjusting of the learning rate has been a subject to many different opinions, some argue that a higher learning rate may "jump over" the global minimum, whilst others argue that lower learning rates may "get stuck" at local minima, failing to reach the overall, global minimum. Through experimentation and a series of trial and error, we deem 0.001 to be the optimal learning rate for this particular task.

We move on to modify the number of features that are extracted from each 2×2 patch. These features are a result of filters in the form of mathematical calculations designed to extract non-linearities from the data. The optimal setup depends on the problem at hand, however we opt to start with the features for each patch as indicated on the TensorFlow website and alter accordingly. Given that the initial model was set with 32 features for each patch, we start by doubling the features to 64, where we see an improvement of 1%. Given this result we continue to double the features until the accuracy saturates at 65% for 256 features for each 2×2 patch. By means of this test we achieve an overall increase of 4% accuracy by increasing the number of features through a series of computations, indicating that CNNs thrive with datasets containing a substantial number of features (such as an image). Furthermore, from an initial 2×2 receptive field setup, we attempt to use a 4×4 patch setup, this however does not improve on the accuracy obtained earlier, as it drops down by 1%.

B. Weekly Dataset

We now move on to evaluate the CNN classifier for the weekly dataset. This dataset was chosen given that it is the second most predictable set with a considerable amount of records. Quarterly and Yearly sets have a small number of examples when compared.

The Weekly dataset is evaluated with the model as described in the methodology, whereby we achieve an overall accuracy of 55%. This result does not match or surpass the 67% obtained by the Logistic Regression classifier. In order to attempt to improve on the 55% accuracy we attempt a number of tests, each time altering the various parameters and settings of the CNN.

We attempt to reduce the complexity of the CNN by removing the second convolutional layer, through which we see accuracy increase substantially to 59%. The second element tweaked was that of the learning rate, the initial model had a learning rate of 0.001. We attempt to higher and lower to learning rate to 0.1 and 0.001, achieving 55% and 54% respectively, both settings not achieving better results than the initial 0.001 learning rate parameter.

We move on to modify the number of features that are extracted from each 2×2 patch. The initial model was set with 32 features for each patch, we start by doubling the features to 64, where we see an improvement of 1% accuracy. Given this result we continue to double the features until the accuracy saturates at 60% for 256 features for each 2×2 patch. From a 2×2 receptive field setup, we evaluate a 4×4 patch setup, this however foes not improve on the accuracy obtained earlier, as it drops down by 2%.

TABLE I: Optimal CNN Setup

Element	Value
Number of Layers	1
Learning Rate	0.001
Dimensions of Receptive Fields	2x2
Feature for each Convolution	256

As one can note, the general flow of the alterations for the weekly dataset are in line with those of the Monthly dataset, whereby the changes alterations that had a positive effect in the monthly dataset, also had a positive effect on the weekly dataset. Our evaluation for the weekly dataset is in line with what was discussed for the monthly dataset, whereby we presume that our problem favours a less complex model with a large number of features. Having said this, for both periodicities we were not able to match or surpass the 69%

and 67% for the monthly and weekly datasets respectively, hence further research is recommended. We tabulate the optimal CNN parameter configuration which applies for both periodicities in Table I.

V. Conclusions

Many deem stock market fluctuations to be random. However, although the stock market is volatile and subject to a vast number of elements, through our experimentation and evaluation we have shown that fluctuations are far from random. With the use of a convolutional neural network, we achieve an accuracy of 65% when forecasting the next month price direction and 60% for the next week price direction forecast. Although other techniques such as Logistic Regression and Support Vector Machines achieve slightly better results in our previous paper, we have shown that deep networks can be configured for the task of financial time series forecasting. Due to the complexity of these deep networks, and sensitivity towards the various parameters, it is possible that by means of further feature engineering and further configuration of network setup, CNNs may outperform the aforementioned state of the art classifiers as was proven in other tasks such as speech and image recognition.

References

[1] Google Brain Team. (February 2017). Deep MNIST for experts. [Online]. Available: https://www.tensorflow.org/versions/r0.9/tutorials/mnist/pros/index.html

[2] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," IEEE, 1998.

[3] Y. LeCun, "Convolutional neural networks (LeNet), deep learning 0.1 documentation," University of Montreal, 2013.

[4] A. Siripurapu, *Convolutional Networks for Stock Trading*, Stanford University Department of Computer Science, 2014

[5] K. H. Kim, *et al.*, "Predicting the success of bank telemarketing using deep convolutional neural network," in *Proc. 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, Fukuoka, 2015.

[6] X. Ding, *et al*., "Deep learning for event-driven stock prediction," in *Proc. the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[7] Y. Tang, *Deep Learning Using Linear Support Vector Machines*, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 2013.

[8] B. Glassman. (February 2017). Currency's impact on your portfolio: Five things you need to know now. [Online]. Available: https://www.forbes.com/sites/advisor/2012/10/03/currencys-impact-on-your-portfolio-five-things-you-need-to-know-now/#71d3bdec5dcb

[9] Investopedia. (January 2017). Index. [Online]. Available: http://www.investopedia.com/terms/i/index.asp

Alexiei Dingli is the head of the Department of Artificial Intelligence within the Faculty of ICT at the University of Malta. He is also one of the founder members of the ACM student chapter in Malta, and a founder member of the Web Science Research, a founder member of the International Game Developers Association (IGDA) Malta and of the Gaming group at the same University. He pursued his PhD on the Semantic Web at the University of Sheffield in the UK under the supervision of Professor Yorick Wilks.

Karl Sant Fournier is a bachelor's degree graduate in business and computing and is a currently a part time student reading for a masters degree with the Department of Artificial Intelligence. He works full time as a business intelligence developer at a local Bank. Given his knowledge of the financial industry he is focusing his thesis on financial time series forecasting, using artificial intelligence techniques.