High Frequency Trading Price Prediction using LSTM Recursive Neural Networks

*

Karol Dzitkowski
k.dzitkowski@gmail.com
Tomasz Kozicki
tom.kozicki@o2.pl
Warsaw University of Technology

Abstract

The Recurrent Neural Network (RNN) is an extremely powerful model for problems in machine learning that require identifying complex dependencies between temporally distant inputs. It is often used in solving NLP problems, therefore it can be suitable also for stock market prediction. In this workwe will try to use recurrent neural network with long short term memory to predict prices in high frequency stock exchange. This paper will show results of implementing such a solution on data from NYSE OpenBook history which allows to recreate the limit order book for any given time.

I. Introduction

Long Short Memory RNN architecture has been proved to be suprisingly successful in sequence prediction resolving a problem with vanishing and exploding gradients. In this work we will use Hochreiter & Schmidhuber (1997) version of LSTM layer. The Long Short-Term Memory (LSTM) is a specific RNN architecture whose design makes it much easier to train. While wildly successful in practice, the LSTM's architecture appears to be ad-hoc so it is not clear if it is optimal, and the significance of its individual components is unclear - which was checked by Google in [1]. The aim of this work is to see if the solution will be suitable for predicting prices in stock markets, which are one of the most difficult time series to predict. Basing on time series data from NYSE OpenBook (which includes every ask and bid prices for one day) we will try to predict next bid or ask value. If there exist any long or short term dependency with historical data, my LSTM model should outperform basic perceptron which will be used for comparison. I will also check if changing LSTM model to GRU (Gated Recurrent Unit - Cho et al. 2014.) will decrease an error rate to establish which solution is best. All technical implementation will be done in Python using Keras [2] library based on Theano. For performance reasons most computations will be done on GPU using NVIDIA CUDA 7.5 technology.

II. Used method

In order to gain some persistent knowledge in the network people invented Recurrent Neural Networks which address this issue. They are networks with loops inside them, allowing information to persist. In that way they are able to learn some long term dependencies between data in several time points. Usually we consider an unrolled RNN with some length

*Project for Computational Intelligence Business Applications Course

creating chain-like structure.

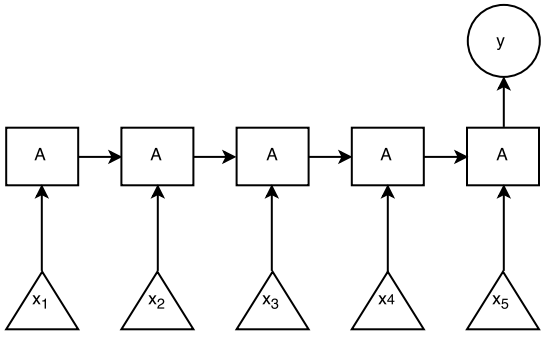

Figure 1: *Recursive Neural Network*

This way they seem to be related to sequences or lists, therefore it will be natural architecture of a network for all time series. In a normal, basic version the hidden layer A is just single *sigmoid* or *tanh* layer. Due to the fact that for learning such an architecture uses Backpropagation Through Time (BPTT) it is usually very difficult to train such networks. The problem is known as vanishing and exploding gradient which makes it impossible for the network to learn long term dependencies. It is solved by using Long Short Term Memory Networks (LSTM) that handles layer A differently. In that model A is more complex structure containing several *tanh* and *sigmoid* layers and +, * operators.

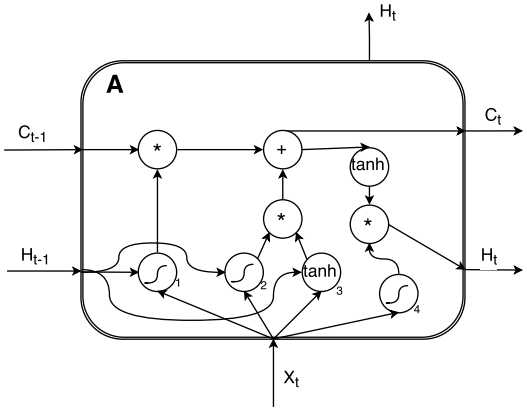

Figure 2: *Recursive Neural Network*

LSTM handles and passes something called the cell state. It can add or remove information to/from the cell and in that way he stores historical data. Mathematical definition of that architecture is shown below, where x_t is our input, H_{t-1} is the previous output and W_m are the weights in layer m:

$$\alpha_t = \sigma(W_1 * x_t + W_1 * H_{t-1} + b_1)$$
$$\beta_t = \sigma(W_2 * x_t + W_2 * H_{t-1} + b_2)$$
$$\gamma_t = tanh(W_3 * x_t + W_3 * H_{t-1} + b_3)$$
$$\theta_t = \sigma(W_4 * x_t + W_4 * H_{t-1} + b_4)$$

then the new cell state and output are:

$$C_t = \alpha_t * C_{t-1} + \beta_t * \gamma_t$$
$$H_t = \theta_t * tanh(C_t)$$

Data from NYSE contains information about the symbol of stock, event type, price and volume as well as very precise time to microseconds. Each record is either bid or ask wchich is indicated by (Side) field of the data.

Table 1: *Selected TAQ NYSE OpenBook Fields*

Symbol	Char(11)
TradingStatus	Char(1)
SourceTime	Int(4)
SourceTimeMicroSecs	Int(2)
PriceScaleCode	Int(1)
PriceNumerator	Int(4)
Volume	Int(4)
Side	Char(1)

For us the most important are price, volume na side data, because we will use them among others as an input to the network. Network will be always trained only for one symbol and for events of submission. We are going to experiment with different inputs, probably using some feature extraction, or choosing custom subset of features by ourselves. Inputs we are going to consider are:

- Time - Time since last ask or bid
- Price - Ask or bid price

- MidPrice - Difference between highest bid and lowest ask price divided by 2
- Volume - Volume of last ask or bid
- Side - Boolean indicating if it was bid or ask
- PriceDiff - Difference of price between last bid or ask

The same inputs will be used to learn ordinary perceptron with one hidden layer. Number of nodes in the hidden layer in that (testing) perceptron will be optimized to maximize its performance. The problem we want to solve is classification - we want to estimate if a price (of ask or bid) will be lower, higher or the same in the next transaction.

For the estimation of performance of my neural networks we will use a mean error loss function:

$$err(t) = \frac{\sum_{i=1}^{t} (y_i \neq Y_i)}{t-1}$$

III. Implementation details

All development will be done using Python language in version 2.7 using Unix system. Keras library will be used for development of the solution, since it provides all necessary layer types including LSTM in the version described above as well as GRU. Since it uses Theano under the hood it is possible to swich on GPU usage, involving that time consuming computations will be executed on my NVIDIA graphic card using CUDA technology.

I. Test data source

As a basic data that our program operates on we have chosen a NYSE OpenBook Ultra data set from 03/04/2013. The data for given stock symbol is parsed from large binary file and can be conveniently saved to the database and/or file. Then this message book is transformed into an order book message by message to capture the transaction price changes in each moment of time, as well as the history of other stock parameters for further building of the features.

II. Algorithms

Classes NeuralNetwork and ClassificationPerformance implements:

- *Feature selection*
  A method of performing feature selection with greedy forward selection using cross validation. We will try to use only selected features in neural network learning procedure. It gets testing data and neural network in parameters and returns a list of numbers of features to use.
- *Crossvalidation*
  A method of using neural network training, cross validating it N times with different test and train data sets randomly choosen from whole dataset. It returns a vector of error rates (training and testing). Data is split in proportion defined by a parameter.
- *T-Student significance test*
  A method *compare* in ClassificationPerformance class uses t-student distance between error rates of two different neural networks, and basing on pvalue is states if there exists significant difference between two solutions. We will use that to determine if really one neural network performs better than the other.

IV. Early Results

Attached is text file „CIBA_earlyResults.txt" which contains execution of the program for $19k$ events of the *AIG* stock order book. Both RNN and MLP report high accuracy from the beginning of iterations. This is very suspicious behaviour that can be explained by the data set characteristics in the next paragraph. As we

can see the cross validation correctly divides the samples for training and testing and the feature selection chooses the best features. Due to no negligible differences in performance the program reports that RNN and MLP are not significantly different.

I. Imbalanced data set

The problem that occurred during test phase turned out to be originating with data approach. As the response value for our vector of features we have choosen indicator wheter the transaction price for a given stock has dropped, stayed the same or risen. This results in response variable being mainly dominated by the second case, because transactions on NYSE OpenBook occur much less often than we initialy expected, making below around 10 percent of overall entries. Above leads to the poor learning performance of the neural networks. It is just easier for them to assume no transaction price change since it is the correct choice most of the time.

V. Reviewed solution

Because we did not get satisfactory results, we had to change our solution in case of features and balance of classes in our data. Ultimately we got balanced data and many more features which led to more reliable results. Below we explain detailed final solution:

I. Data set

As message book in it's raw form yields little indication of transaction price changes there is need to convert it into more structured form. The representative structure that groups prices and volumes on both buyers and sellers side is called order book. Each event from the message book regarding analyzed stock symbol is compared with existing order book records. These are grouped into two lists: bid side and ask side. The bid side is sorted descending by price and at each price level contains the accumulated volume information. The ask side is sorted ascending by price and at each price level contains the accumulated volume information. Whether it is buy or sell order it is checked if the price allows for a transaction to take place. If so the current transaction price is updated while previous transaction price still being held for comparison. Otherwise this message is added to appropriate record in order book with matching price (total volume is incremented).

II. Features and response class

We identify a collection of proposed attributes that are divided into two categories: basic and time-insensitive, all of which can be directly computed from the data. Attributes in the basic set (v1 in the Figure 3) are the prices and volumes at both ask and bid sides up to n different levels (that is, price levels in the order book at a given moment), which can be directly fetched from the order book. Attributes in the time-insensitive set are easily computed from the basic set at a single point in time. Of this, bid-ask spread and mid-price, price ranges, as well as average price and volume at different price levels are calculated in feature sets v2, v3, and v4, respectively; while v5 is designed to track the accumulated differences of price and volume between ask and bid sides.

With each new message arriving to the order book the history of features and output is appended with new set of those. The response variable is a category number indicating whether the transaction price for a given stock has dropped, stayed the same or risen in the next moment.

Basic Set	Description(i = *level index*)
$v_1 = \{a_i^k\}_{i=1}^{4n} = \{P_i^{ask}, V_i^{ask}, P_i^{bid}, V_i^{bid}\}_{i=1}^{n}$,	*price and volume(n levels)*
Time-insensitive Set	Description(i = *level index*)
$v_2 = \{a_k^5\}_{k=1}^{2n} = \{(P_i^{ask} - P_i^{bid}), (P_i^{ask} + P_i^{bid})/2\}_{i=1}^{n}$,	*bid-ask spreads and mid-prices*
$v_3 = \{a_k^5\}_{k=1}^{2n} = \{\lvert P_{i+1}^{ask} - P_i^{ask}\rvert, \lvert P_{i+1}^{bid} - P_i^{bid}\rvert\}_{i=1}^{n}$,	*price differences*
$v_4 = \{a_k^5\}_{k=1}^{4} = \{\frac{1}{n}\sum_{i=1}^{n} P_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} P_i^{bid}, \frac{1}{n}\sum_{i=1}^{n} V_i^{ask}, \frac{1}{n}\sum_{i=1}^{n} V_i^{bid}\}$,	*mean prices and volumes*
$v_5 = \{a_5^5\}_{k=1}^{2} = \{\sum_{i=1}^{n}(P_i^{ask} - P_i^{bid}), \sum_{i=1}^{n}(V_i^{ask} - V_i^{bid})\}$,	*accumulated differences*

Figure 3: *Feature sets*

As depicted in the table above, the level variable controlled by program describes how deep the order book we want features to rep-

resent. The higher the level the more distant rows from the cheapest ask and most expensive bid are taken into account. Therefore, overall number of features is result of the formula: 4 * level + 2 * level + 2 * level + 4 + 2.

As the number of features rises with incremented level we considered that it may be beneficial for computation time to select features which are most important. If a number of them would stand out in terms of effect on accuracy then it would be wise to choose thiem as a best subset for training. To evaluate this aspect we used mentioned feature selection algorithm. For this the initial feature set based on level equal to 3 was determined to 30. As it turned out there is no significant difference in impact of features on accuracy which is shown on the barplot below. Therefore we decided to conduct the final evaluation on all initial features.

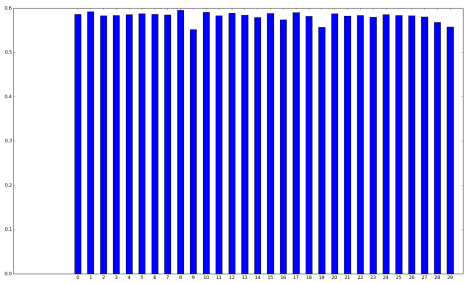

Figure 4: *Features importance*

III. Input and output form

Before it is passed to neural network each order book history record consists of n consecutive records from order book history. Each of this records consists of all feature values and response variable. This data is passed to the neural networks in the following form: history of feature vector from input length last events (window size) and a single response variable indicating price change after the next event. The neural network recieves batch of these records with it being balanced in terms of response variable classes. Balancing the number of NN input records is achieved by taking the class which occurs least frequently, adjusting other class records count to this number and shuffling the combined features history-response class set.

IV. Networks

At the end we resigned from using GRU network and instead focused on LSTM, comparing it to ordinary MLP. For learning we used Stochastic gradient descent optimizer with categorical crossentropy loss function. Below we present actual models of networks used:

```
PERCEPTRON:
model = Sequential()
model.add(TimeDistributedDense(output_dim=self.hidden_cnt,
                               input_dim=self.input_dim,
                               input_length=self.input_length,
                               activation='sigmoid'))
model.add(TimeDistributedMerge(mode='ave'))
model.add(Dropout(0.5))
model.add(Dense(self.hidden_cnt, activation='tanh'))
model.add(Dense(self.output_dim, activation='softmax'))

LSTM:
model = Sequential()
model.add(LSTM(output_dim=self.hidden_cnt,
                          input_dim=self.input_dim,
                          input_length=self.input_length,
                          return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(self.hidden_cnt, activation='tanh'))
model.add(Dense(self.output_dim, activation='softmax'))
```

V. GPU usage

Because of the fact that our NN are learning slowly and teaching them is time consuming we found useful to switch on BLAS and CUDA enabled devices usage by Theano library. To do that it was sufficient to create a file /.theanorc with content:

```
[global]
floatX = float32
device = gpu1

[blas]
ldflags = -L/usr/local/lib -lopenblas

[nvcc]
fastmath = True
```

Using that new architecture and solution we received our final results.

VI. Final Results

The program allows tuning of following parameters: number of iterations, features level, input length (window size), hidden layer neurons count. Based on adjusting these parameters series of test runs were executed.

As the test data we have chosen AIG order book generated by analysing 2 milion records from NYSE OpenBook. This yielded a balanced test data set in size of 33k.
33120 Total records including:
11040.0 records with price down
11040.0 records with price stable
11040.0 records with price up

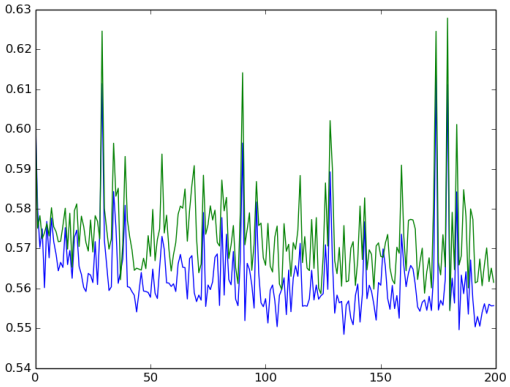

Figure 5: *RNN - 200 iterations, level = 2, input length = 25, 50 neurons*

In the plot above the train error oscillates between 0.55 and 0.57. The test error oscillates around 0.58 and 0.56.

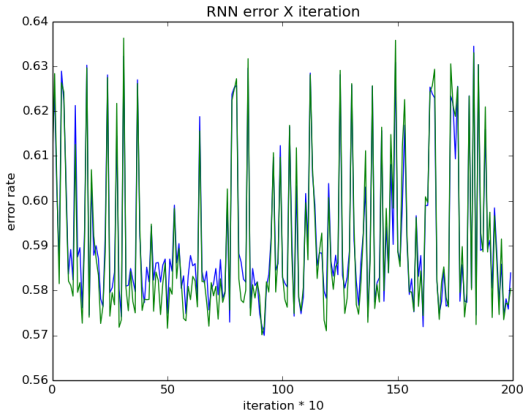

Figure 6: *RNN - 200 iterations, level = 6, input length = 25, 50 neurons*

In the plot above there are large oscillations for both train and test error, which range basically from 0.57 to 0.63.

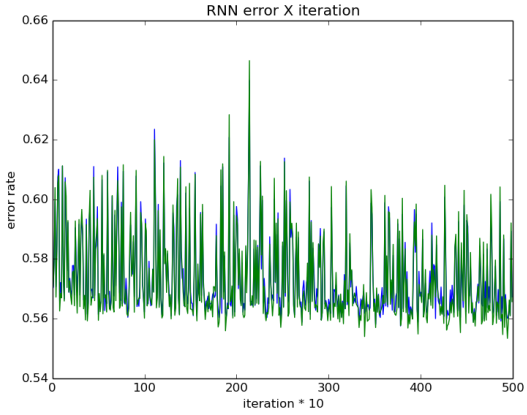

Figure 7: *RNN - 500 iterations, level = 3, input length = 50, 100 neurons*

In the plot above there are large oscillations for both train and test error, which range basically from 0.56 to 0.60.

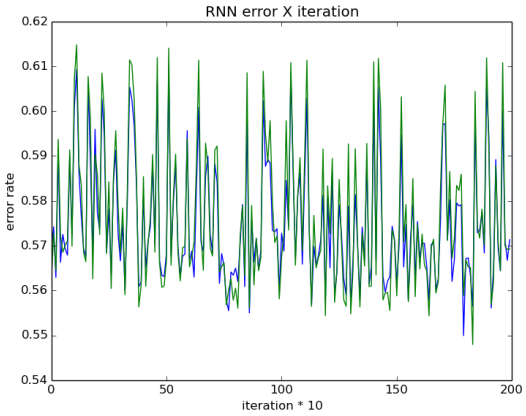

Figure 8: *RNN - 200 iterations, level = 4, input length = 100, 100 neurons*

In the plot above there are large oscillations for both train and test error, which range basically from 0.56 to 0.61.

Tuning of paramaters showed in the following to be the most productive for our final comparison of RNN:

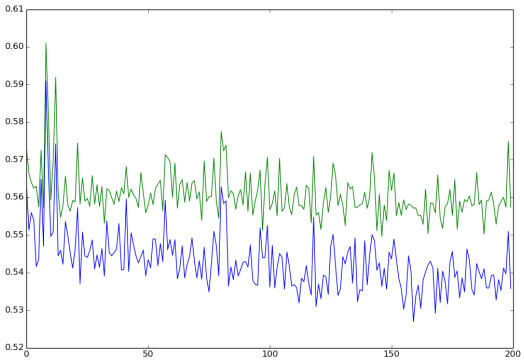

Figure 9: *RNN - 200 iterations, level = 4, input length = 25, 50 neurons*

With train error going as down as 0.53 and quite stable test error 0.56, this is the best neural network configuration we have found so far. So we decided to compare it with MLP using the same parameters.

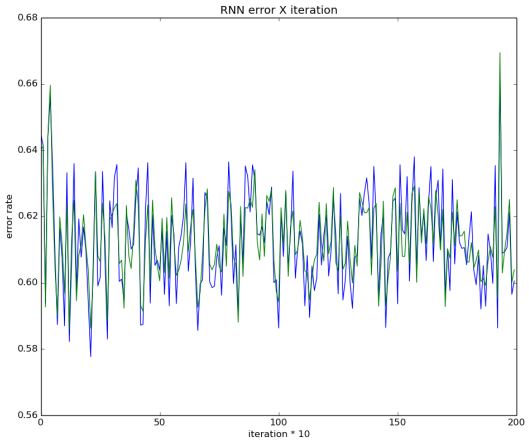

Figure 10: *MLP - 200 iterations, level = 4, input length = 25, 50 neurons*

In the plot above created for MLP there are large oscillations for both train and test error, which range basically from 0.59 to 0.63.

Based on this runs the boxplot was created

to show how statistically different error rates of these two neural networks are.

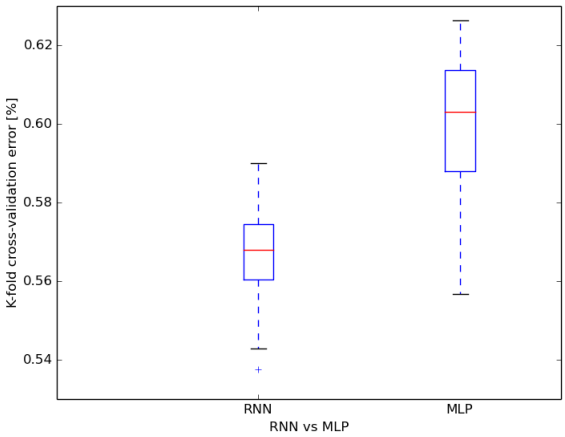

Figure 11: *Error rate boxplot*

The boxplot confirms that there is distinguishable difference in favor of RNN.

Listing 1: *RNN vs MLP with 200 epochs and 20 cross-validation steps output:*

```
RNN is significantly better than MLP
RNN avg err = 0.567300724638,
MLP avg err = 0.600966183575
```

VII. Conclusion

The research showed that the RNN with LSTM layer performs better than ordinary MLP. Although the difference is much smaller that was expected from fundamental neural network philosophy that divides these two approches. For us the most important value is learning value that results from solution spanning over many topics like:

- Order Book mechanics
- RNN LSTM structure and purpose
- Python Keras library usage with GPU-accelerated computations
- Feature selection algorithm
- Crossvalidation algorithm
- T-Student significance test
- Data set balancing (subsampling)

Further work on this topic should include experimenting with detail of neural networks parts like optimizers, loss functions, layer types. It is worth to further investigate how parameters such as number of iterations, level, input length, neurons count can be optimize to achieve better overall results for both neural networks types. Other research papers also suggest using mid-price change instead of last completed transaction change as response variable.

References

[1] "Modeling high-frequency limit order book dynamics with support vector machines"
http://www.math.fsu.edu/ aluffi/archive/paper462.pdf
Alec N. Kercheval and Yuan Zhang, 24 October 2013

[2] "An Empirical Exploration of Recurrent Network Architectures"
http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf
Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever

[3] "Long short-term memory"
http://deeplearning.cs.cmu.edu/pdfs/Hochreiter97_lstm.pdf
Sepp Hochreiter and Jurgen Schmidhuber, 1997

[4] "LSTM implementation explained"
https://apaszke.github.io/lstm-explained.html
Adam Paszke, 30 Aug 2015

[5] "Recurrent Neural Networks Tutorial - Implementing a RNN with Python, Numpy and Theano"
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/

[6] "Keras library"
http://keras.io

[7] "Supervised Sequence Labelling with Recurrent Neural Networks" *http://www.cs.toronto.edu/graves/preprint.pdf* Alex Graves