High Frequency Trading Price Prediction using LSTM Recursive Neural Networks

*

Karol Dzitkowski
k.dzitkowski@gmail.com
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 work I 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 successive in sequence prediction resolving a problem with vanishing and exploding gradients. In this work I 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) I 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 creating chain-like structure.

*Project for Computational Intelligence Business Applications Course

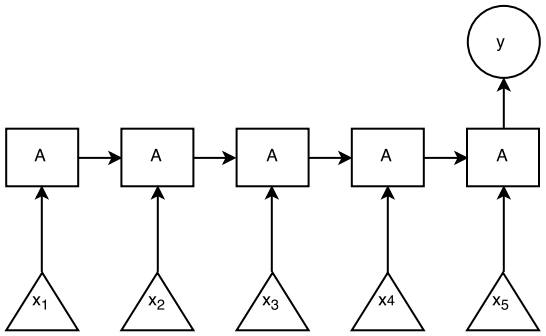

Figure 1: *Recursive Neural Network*

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.

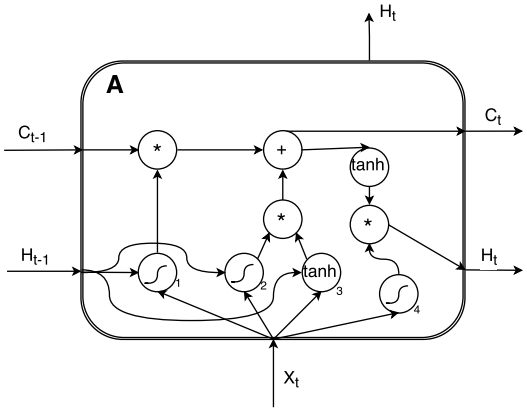

Figure 2: *Recursive Neural Network*

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 me the most important are price, volume na side data, because I 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. I am going to experiment with different inputs, probably using some feature extraction, or choosing custom subset of features by myself. Inputs I am 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 output for both neural networks will be the next bid or ask price which should appear in the book.

For the estimation of performance of my neural networks I will use mean squared error loss function:

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

III. Implementation details

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.

IV. Results

V. Conclusion

References

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

[2] "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/*

[3] "Keras library" - *http://keras.io*

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