2018 International Conference on Computing, Power and Communication Technologies (GUCON)
Galgotias University, Greater Noida, UP, India. Sep 28-29, 2018

Forecasting Time Series Stock Data using Deep Learning Technique in a Distributed Computing Environment

Lasani Hussain
High Performance Computing Lab,
School of Computer Science and Engineering
National Institute of Science & Technology, Berhampur, India,761008

Sekhar Banarjee
High Performance Computing Lab,
School of Computer Science and Engineering
National Institute of Science & Technology, Berhampur, India,761008

Sumit Kumar
High Performance Computing Lab,
School of Computer Science and Engineering
National Institute of Science & Technology, Berhampur, India,761008

Aditya Chaubey
High Performance Computing Lab,
School of Computer Science and Engineering
National Institute of Science & Technology, Berhampur, India,761008

Motahar Reza
High Performance Computing Lab,
School of Computer Science and Engineering
National Institute of Science & Technology, Berhampur, India,761008
reza@nist.edu

***Abstract*— Recurrent neural networks (RNN's) are a variant of artificial neural networks (ANN's) that perform well on nonlinear and time-series data forecasting. It was discovered that vanilla RNN finds it hard to learn long-term dependencies. LSTM and GRU are modifications that overcome this problem. We propose a new distributed computing model in this paper which used on time-series stock data. Our study attempts to provide a comprehensive and objective assessment and later we do a comparison of performance of three models namely vanilla RNN, LSTM and GRU in a distributed cluster. This paper offers practical insights and potentially useful directions for further research into how Spark cluster can be used to reduce training time by using it on top of existing machine learning models.**

Keyword— Forecasting, Stock price, Deep RNN, Distributed Processing, LSTM, Spark , GRU.

I. INTRODUCTION

Developing a mathematical model to simulate non-linear relationships between input and output parameters is a difficult task as relationship between input and output may be complicated. Stockbrokers mostly predict stock trade by using time series based methods for prediction. However usually these methods don't guarantee accurate predictions because these rely on previous trends and do not consider the fluctuations in the price history. Hence we try to explore improvised methods of prediction. The studies of Schumaker and Hsinchun[1] contradict the efficient market hypothesis. RNNs are a special case of neural networks that exploit the sequential nature of time–series data by assigning equal weights across multiple time steps [2, 3]. Recurrent Neural nets can theoretically remember the previous data and based on that can make the prediction. Embedded in this idea is a belief that the stock ticker has some sort of periodic pattern. Global events happening across the world have both long and short term effect on prices of stock as well as their periodic variation. One hopes that the RNN would learn these patterns and be able to predict future prices based on the information content of what was learned. The RNN architecture allows for cycles in neurons, allowing them to have a form of memory not available in other architecture. RNN's can be thought of as ANN's hidden layers folded in time [4]. This makes it possible for them to recognize temporal effects. Unlike ANN's or other learning algorithms, RNNs can automatically extract relevant features from the feature set [3,5]. However vanilla RNN suffer from problems like vanishing and exploding gradient while back propagating the error a large number of times. Long Short-Term Memory units, or LSTMs, are added to vanilla RNN's which effectively handles the vanishing gradient problem. This study tackles these issues by adopting RNN along with individual and separate implementations of LSTM and GRU both on a sequential and distributed computing environment for its exceptional performance.

II. MOTIVATION AND CONTRIBUTION

It is true that the methods mentioned above have successfully implemented stock-prediction but there are still many shortcomings like the vanishing and exploding gradient problem. The Simple Recurrent Network (SRN) was conceived and first used by Jeff Elman, and was first published in a paper entitled Finding structure in time. Non-linear models overcome the limitation of linear models by finding long tern dependency patterns in data [6].Chen, Fan, Chen, and Wei (2013) used ANN for optimizing parameters [7]. Kim and Ahn conducted further research to find global optimization approach for problems in which ANN's performed poorly [8]. Hsu proposed a model by combining the features of back propagation, feature pruning and genetic algorithms to tackle price forecasting problems [9].

III. LSTM AND GRU ARCHITECTURE

Recurrent networks take as their input both the current input at time step t as well as the information learned from data at time step t-1. Basically RNN's take input from the present and the recent past, which together determine the output at current time step. A recurrent neural network contains multiple copies of the same network which feeds the output back to the input [4]. Consider what happens if we unroll the loop:

An illustrative example of a recurrent neural network which has one hidden layer and stretched out in time over a chronological succession of six time steps. Each input X_t is in the succession of X_{t-1}. The hidden layer has H units and the j^{th} output at time step t is denoted by h_{jt}. The connections between the hidden units are repeated and are weighted by the matrix W_h . At the last time step t, the hidden units connect to a soft-max layer 1-of-K output vector Y_t.

Sequential dependency pattern is preserved in the recurrent network's obliterated state, which deals to bridge many time steps as it showers forward to involve the processing of each new example. We depict the procedure of conveying memory forward mathematically:

$$h_t = \phi(Wx_t + Uh_{t-1}),$$

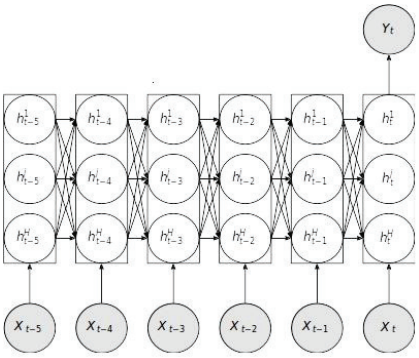

Fig. 1. Unrolling a RNN

The concealed state at time step t is h_t. It is a mapping of the input at the same time step x_t, altered by a weight matrix W added to the concealed state of the old time step h_{t-1} multiplied by its own concealed-state-to-concealed-state matrix U, also known as a transition matrix and similar to a Markov chain.

RNN's are good at learning patterns where there are short term dependencies. But there are also cases where we need more context. RNN's suffer from vanishing/exploding gradient problem in situations where there are long term dependencies. This problem was researched rigorously by Hochreiter (1991) [German] and Bengio, et al. (1994),

Sepp Hochreiter and Juergen Schmidhuber [10] suggested a version of vanilla RNN's with Long-Short-Tern-Memory units. LSTMs solve to the long-term dependency problem.

LSTMs contain information in a gated cell. These gates apply multiplication and use sigmoid as squashing function.The cells learn when to allow/discard/delete the data through iterative guesses, back propagating errors and adjusting weights.

We use a neural network to learn the remember gate which decides which long-term memory units to continue remembering and which to forget which depends on the backpropagation from comparing the final output with our original labels:

$$remember_t = \sigma(W_r x_t + U_r\, wm_{t-1})$$

We also compute the information we can learn from x_t , i.e.,

$$ltm'_t = \phi(W_l\, x_t + U_l wm_{t-1})$$

ϕ is an activation function which squashes the output to a given required range (out of sigmoid, relu etc. we choose tanh). We selectively remember only those units which contribute to our learning: $save_t = \sigma(W_s\, x_t + U_s\, wm_{t-1})$. After discarding neuron units which we won't need again and saving practical pieces of information, the long-term memory is:

$$ltm_t = remember_t \circ ltm_{t-1} + save_t \circ ltm'_t$$

where $\circ$ denotes element-wise multiplication. We learn a focus/attention vector which stores information from LSTMs that is instantly used: $focus_t = \sigma\,(W_f\, x_t + U_f\, wm_{t-1})$ Our working memory is then $wm_t = focus_t \circ \phi(ltm_t)$ The Fig 2 summarizes the above:

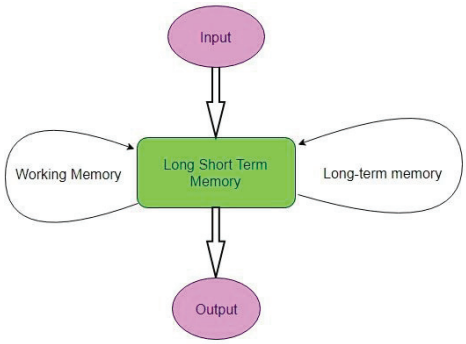

Fig. 2. Schematic diagram of LSTM

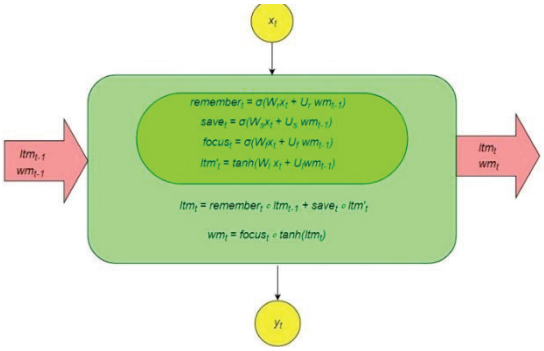

Fig. 3. Equations of LSTM

The gates for a GRU cell are illustrated in Fig Fig 4. GRU replaces the forget and input gates in the LSTM unit by an update gate and reset gate. The update gate determines the amount of old memory to retain and the reset gate determines the aggregation of fresh input with the old memory. The following equations define the gating mechanism in a GRU

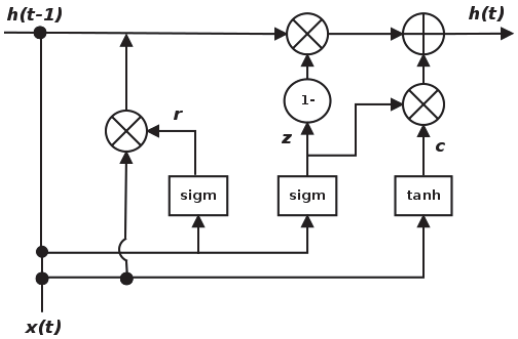

Fig. 4. GRU architecture

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$

$$r = \sigma(W_r h_{t-1} + U_r x_t)$$

$$c = tanh(W_c(h_{t-1} \otimes r) + U_c x_t)$$

$$h_t = (z \otimes c) + \big((1 - z) \otimes h_{t-1}\big)$$

IV. SPARK ARCHITECTURE

To process petabytes of data in a distributed computing environment Hadoop was developed and open sourced, taking its inspiration from the Google File System (GFS) and the adjoining distributed computing framework, MapReduce. Apache Spark is an advancement on the original Hadoop MapReduce. Since Spark does in-memory computation it is approximately 10 to 100 times faster than Hadoop and is well suited for streaming data.

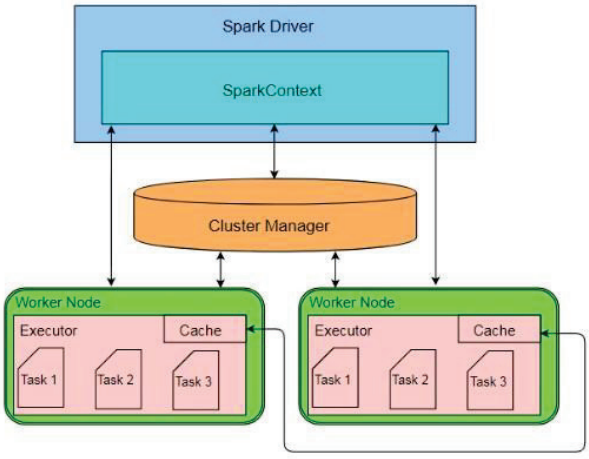

Fig. 5. Architecture of Spark cluster

Spark (is written in Scala, and therefore integrates natively with the Java Virtual Machine (JVM) powered ecosystem. Spark had early on provided Python API and bindings by enabling PySpark.

V. TEXPERIMENTS RESULTS

The experimental environment is: Spark cluster was used which consists with 10 machines of core i5 processor memory given was 4 GB The methods selected to offer comparison results include: LSTM, GRU. In this study, we have used the stock market dataset which can be obtained from https://www.kaggle.com/dgawlik/nyse.csv. The dataset contains information from 2010 to the end of 2016 on daily basis. Only the stock price of Yahoo has been used to train the dataset

TABLE I. PERFORMANCE ANALYSIS FOR VANILLA NEURAL NETWORK WITH DIFFERENT NUMBER OF NODES IN HIDDEN LAYER

Configuration	Layers	Units	RMSE
1.	1	15	0.683
2.	1	20	0.611
3.	2	40	0.645
4.	2	15	0.639
5.	2	40	0.602
6.	3	15	0.613
7.	3	15	0.598
8.	3	20	0.584

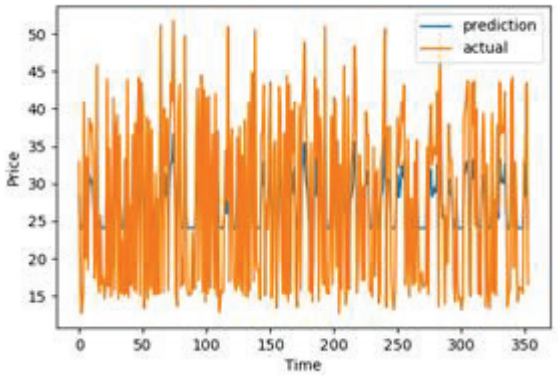

Fig. 6. Vanilla RNN Actual vs predicted value

Hyper parameter tuning was done by distributing the 8 different configurations of each model on a 8 node spark cluster and it was found that 3 layered/20 units configuration gives the best result and thus this architecture is used for further analysis.

VI. PERFORMANCE ANALYSIS

A. Training Phase

1) Vanilla RNN:

* For back propagation Adam Optimizer has been used with learning rate 0.001 and trained on 1000 epochs.

2) LSTM network:

* RMSE cost function was used and Adam Optimizer for back propagation

TABLE II. PERFORMANCE ANALYSIS FOR LSTM WITH DIFFERENT NUMBER OF NODES IN HIDDEN LAYER

Configuration	Layers	Units	RMSE
1.	1	15	0.750
2.	1	20	0.741
3.	2	40	0.726
4.	2	15	0.738
5.	2	40	0.710
6.	3	15	0.713
7.	3	15	0.701
8.	3	20	0.628

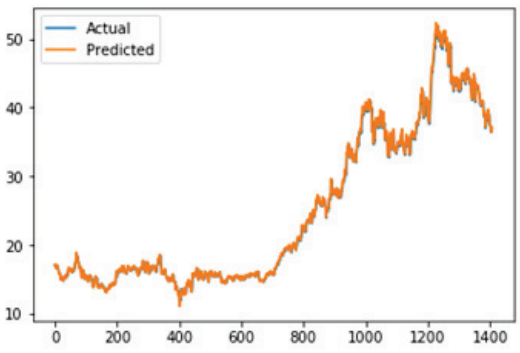

Fig. 7. LSTM actual value vs predicted value

3) GRU network:

* RMSE cost function was used and Adam Optimizer for back propagation

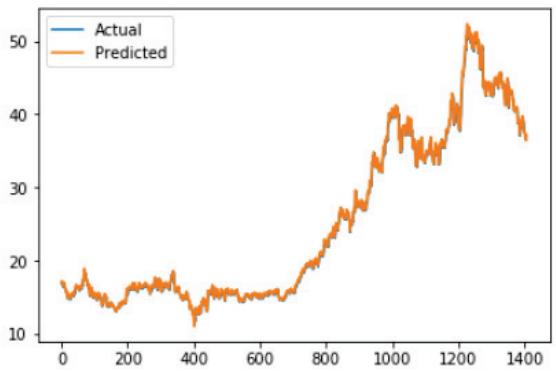

Fig. 8. GRU Actual vs predicted value

B. *Testing Phase*

We observe that the error during training phase is almost negligible for both LSTM and GRU. This can be observed by analysing the difference between red and blue lines indicating predicted and actual values respectively in Figure 7 and Figure 8.

TABLE III. PERFORMANCE ANALYSIS FOR GRU WITH DIFFERENT NUMBER OF NODES IN HIDDEN LAYER

Configuration	Layers	Units	RMSE
1.	1	15	0.579
2.	1	20	0.561
3.	2	40	0.551
4.	2	15	0.548
5.	2	40	0.512
6.	3	15	0.538
7.	3	15	0.489
8.	3	20	0.472

TABLE IV. COMPARISON ANALYSIS : VANILLA RNN, LSTM AND GRU TRAINING TIME IN SECONDS

Model	Vanilla RNN	LSTM	GRU
Sequential	358.6	289.3	263.1
3 nodes Spark cluster	293.2	208.4	183.8

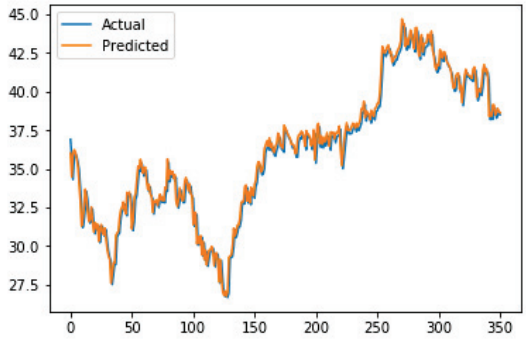

Fig. 9. LSTM actual value vs predicted value

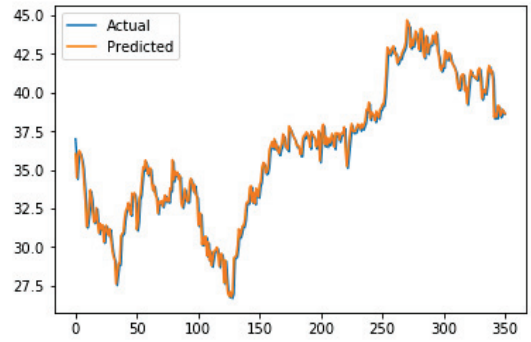

Fig. 10. GRU Actual value vs predicted value

By comparing the RMSE scores in Tables-I, II and III we can clearly see that a 3 layer/20 node architecture gives the best performance in all three cases of RNN,LSTM and GRU.

* Between LSTM and GRU, GRU has less training time because of its simpler architecture.

* From figures 9 & 10 we observe GRU performs slightly better than LSTM.

* Vanilla RNN takes 28% less time for training on a 3 node Spark cluster than on single node

*LSTM takes 27% less time for training on a 3 node Spark cluster than on single node

* GRU takes 30% less time for training on a 3 node Spark cluster than on single node

* Training time on a Spark cluster is significantly less than that on a single node.

* This model might be improved by using window size greater than 1 and Deep Q learning can also be applied in this model

VII. CONCLUSION

The experiments demonstrate the robustness and effectiveness of the proposed method. In the meantime, these work clearly manifested that the implementation of GRU and LSTM for RNN on Spark cluster offered a significant improvement for prediction of stock data. RNN has proved to be a powerful tool for learning long-term dependencies and LSTM and GRU have emerged as two powerful modifications to control vanishing gradient problem. Also we observed that Spark provided an average of 1.3x speedup on training time on our data. Thus how to combine their advantages to explore and apply machine learning models on big data is an interesting and essential research topic.

REFERENCES

[1] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. ACM Transactions on Information Systems (TOIS), 27(2):12, 2009.

[2] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In NIPS, volume 400,page 409. Citeseer, 1995.

[3] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[4] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In Advances in Neural Information Processing Systems, pages 190–198, 2013.

[5] S. C. Prasad and P. Prasad. Deep recurrent neural networks for time series prediction. arXiv preprint arXiv:1407.5949, 2014.

[6] Armano, G., Marchesi, M., & Murru, A. (2005). A hybrid genetic-neural architecture for stock indexes forecasting. Information Sciences, 170, 3–33.

[7] Chen, M., Fan, M., Chen, Y., & Wei, H. (2013). Design of experiments on neural network's parameters optimization for time series forecasting in stock markets. Neural Network World, 23, 369–393.

[8] Kim, K., & Ahn, H. (2012). Simultaneous optimization of artificial neural networks for financial forecasting. Applied Intelligence, 36, 887–898.

[9] Hsu, V. (2013). A hybrid procedure with feature selection for resolving stock/futures price forecasting problems. Neural Computing and Applications, 22, 651–671.

[10] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Juergen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, eds., A Field Guide to Dynamical Recurrent Neural Networks.