(IJAER) 2020, Vol. No. 20, Issue No. IV, October **e-ISSN: 2231-5152, p-ISSN: 2454-1796**

USING LSTM BASED RNN ON TIME SERIES DATA TO PREDICT STOCK INDEX VALUES AND CLASSIFY ASSET PRICE MOVEMENTS

Lavaneesh Sharma

ABSTRACT:

We propose designing and training RNNs for univariate time series data to forecast future values on S&P 500 data. Firstly, time series data will be prepared, within a single LSTM layer framework that will help to predict stock index values. Secondly, we will build a deep RNN with three inputs to classify asset price movements. Lastly, we will combine 2 layer, stacked LSTM with learned embedding's and one-hot encoded categorical data. The paper also gives a brief overview of Recurrent Neural Networks and their popular architectures currently in use.

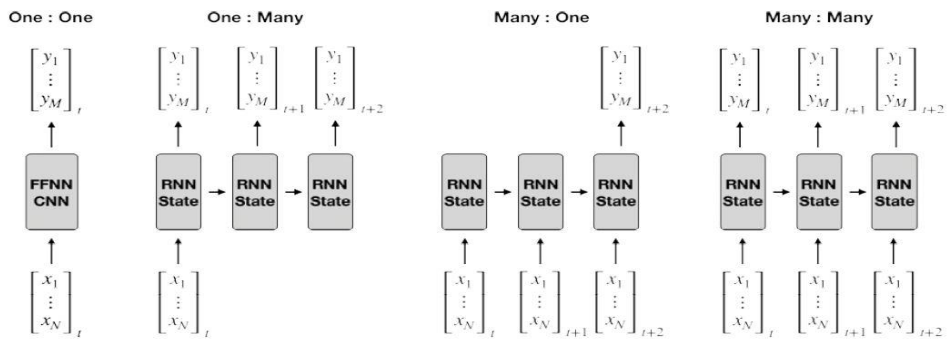
1. INTRODUCTION:

The conventional neural network like Feedforward Neural Networks treat every sample feature vector as identically distributed and independent. As such they don't incorporate previous data samples, while evaluating current observation and are memoryless. Recurrent Neural Networks on the other hand are memory based neural networks. What truly sets them apart from CNN's and other neural networks is that each output is obtained using both previous information and new one, which, in recurrent format, when used for given set of fixed computations, enables a parameter sharing across deeper computational graph.

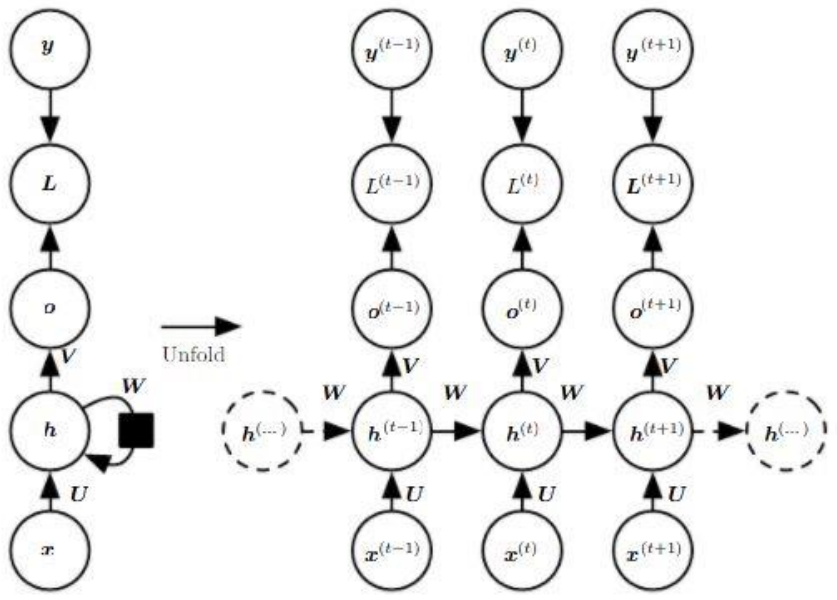
1.1 Neural Networks: Neural Networks are supervised type Machine Learning algorithm that imitate the functioning of brain cells or neurons. The supervised Machine learning algorithms are used to generate or predict a set of labels for new data, based on learnings from a fixed set of previous data. The output can be Continuous (Regression) or Discrete (Classification).

The associations of the organic neuron are displayed as weights. A negative weight signifies inhibitory associations; while positive weight mirrors an excitatory association. All data sources are changed by a weight and added, in the form of a linear combination. At last, an activation function controls the output's amplitude. For instance, a satisfactory result is normally somewhere in the range of 0 and 1, or it very well may be ± 1.

1.2 Recurrent Neural Networks: RNNs are neural networks for successive information, thusly we apply them to time series. The fundamental thought behind repetitive neural organizations is utilizing not just current inputs, yet in addition, the past data, for making the current forecast. This thought bodes well as we could fabricate neural networks passing qualities forward as expected. Although, such straightforward arrangements generally don't fill in true to form. They are difficult to prepare and might perform as expect. Or maybe, we have to have a framework with some sort of memory. RNNs find their use in sequential data wherein we find patterns evolving over time and the learning usually incorporates memory of previous data-points.



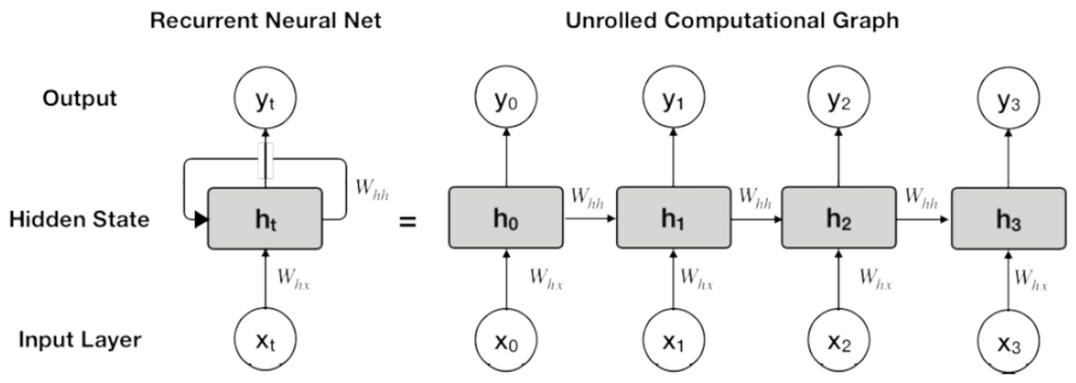
Different Sequence to Sequence models



The computational graph to compute the training loss of a recurrent network that maps an input sequence of ***x*** values to a corresponding sequence of output ***o*** values. RNNs are called recurrent because they apply the same transformations to every element of a sequence in a way that the output depends on the outcome of prior iterations. As a result, RNNs maintain an internal state that captures information about previous elements in the sequence akin to a memory.

RNN's maintain an **internal state** capturing information about previous items. A baseline model or a **computational graph (uncontrolled)** is shown below that has two weight matrix's given by Whh and Whx applicable to previous and current inputs respectively. The output is given by y_t which is obtained by matrix multiplication and the resultant passed through an activation function like $ReLU$ or $\tanh x$

$$y_t = g(W_{hh}h_{t-1} + W_{xh}x_t)$$

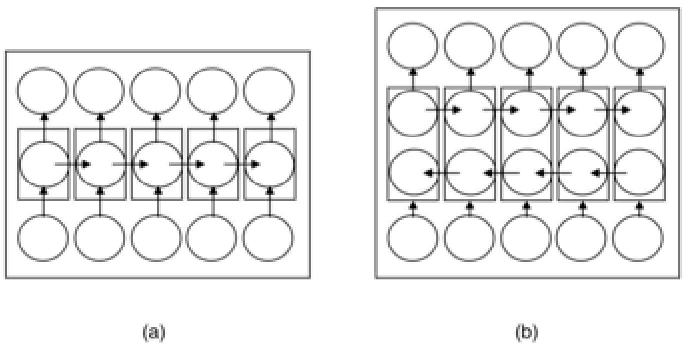


1.3 Backpropagation algorithm is carried out from right to left (output to input side), post updating the weights in the forward pass from left to right i.e. input to output vis-à-vis unrolled computational graph. It constantly evaluates a loss function and calculates its gradient (∇) w.r.t. parameters to update weights. Hence the name, backpropagation.

2. RNN ARCHITECTURE REVIEW

1. **Output Recurrence and teacher forcing:** The computational difficulty of hidden state recurrences stems from connecting a unit's hidden state to itself. This can be eased by connecting it to the output directly. As such, it has capacity w.r.t baseline model decreases but, parallel training of decoupled time steps is possible. But, for effective learning, it is also imperative that the train samples reflect this information for backpropagation to update weights consequently. Financial data usually doesn't meet this requirement as asset returns are usually uncorrelated to previous values. The simultaneous integration of prior output and input vectors in called teacher forcing.

2. **Bi-Directional RNN:** It can sometimes be relevant that output may rely on future elements too. As such Bi-directional RNN's are known to link 2 hidden layers of opposite-directions, to same output. Therefore, we can obtain such a network, by splitting a regular RNN's neurons into 2 directions, in such a way that, the output layer can get information backwards (from past) and forward (from future) states simultaneously. General structures are depicted as:



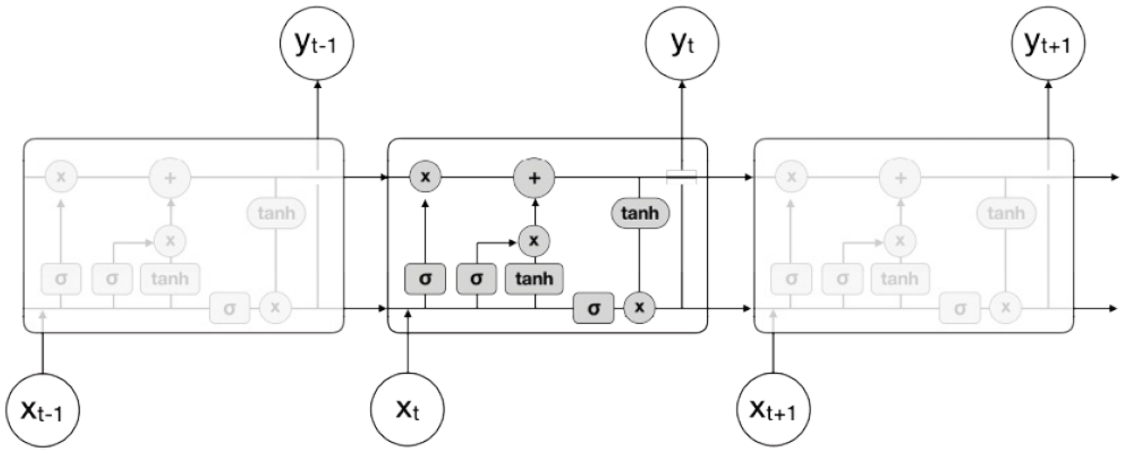
Structure overview
(a) unidirectional RNN
(b) bidirectional RNN

3. **Encoder-Decoder architecture, attention and transformers:** Encoder-decoder architectures allow for equal length input & output sequences. The **encoder** is a type of RNN that maps input vector into, an entirely different space – called latent space; while the **decoder** function is a complementary RNN that maps the encoded input to the target space. Both of them are incorporated together in the training process such that the input vector of last encoder hidden state becomes an input to decoder, which can learn to match training samples. The **attention mechanism** tends to a restriction of utilizing fixed-size encoder inputs when info arrangements themselves fluctuate in size. The instrument changes over crude content information into an appropriated portrayal stores the outcome, and utilizations a weighted normal of these element vectors as setting. The weights are found out by the model and switch back and forth between putting more weight or consideration regarding unique components of the information. An ongoing **transformer** design forgoes repeat and convolutions and solely depends on this consideration instrument to learn input-yield mappings. It has accomplished prevalent quality on machine interpretation errands while requiring considerably less time for preparing, not least since it very well may be parallelized.

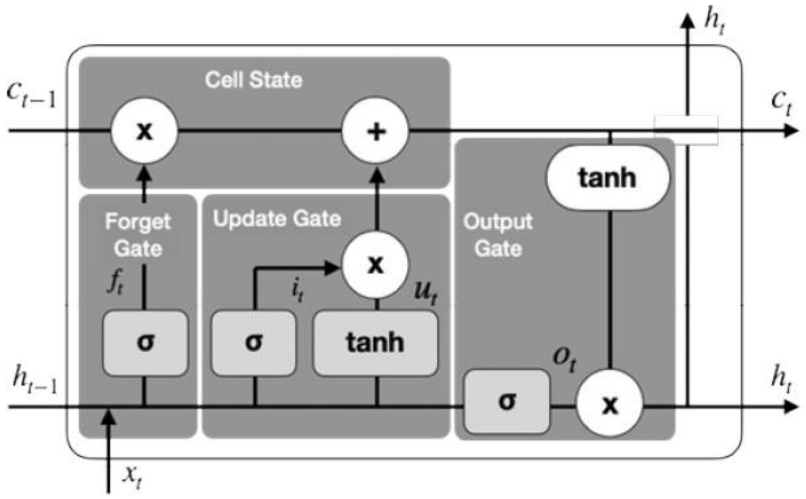
Adding more layers or increasing the depth of FNNs or CNNs allows them to take on more complex learning problems. Likewise, RNNs benefit from decomposing input-output mapping to multi-layers. **Stacking recurrent layers** is a typical way (for regression and classification tasks), which allows for lower layers to encapsulate high frequency data, processed by a higher layer into low frequency characteristics. Conceptually, RNNs, as such can make use of such long sequences. However, in reality, they toil to derive useful context information from distant past. Frequent multiplication whilst, Back-prop, makes the gradient vanish or infinite. It has been shown that SGD shows serious issues in training 10 to 20 elements RNN network. The solution has been presented in the form of **leaky units** or **echo state networks.** Skipping time steps or integrating different frequency signals are yet other proposed solutions. But the best proposed solution is using gated networks that keep tracks of (1) current state and (2) when to forget the data, which allows learning back to hundreds of time steps behind. The most popular are as follows:

4. **LSTM or Long Short Term Memory**: Long short-term memory is a gated memory unit for neural networks. It utilizes gates that manage memory contents and elements of input sequence.

The gates are logistic functions of weighted sums, where back prop helps to find weights. It can learn, remember or recall, without any special training. A cell, an input gate, an output gate and a forget gate are typical components of LSTM network. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. The information flow in an LSTM cell is shown as:



The transformation and passing of vectors enables the interaction of four parameterized layers. Layers consist of (1) Input gate (2) Output gate (3) forget gate. The white cells signify pair-wise operations, while layers are shown by grey cells as shown below:



The input gate (1) and the forget gate (2) manage the cell state (4), which is the long-term memory. The output gate (3) produces the output vector or hidden state (5), which is the memory focused for use. This memory system enables the network to remember for a long time, which was badly missing from vanilla recurrent neural networks.

LSTM Equations are given as:

$$f_t = sigmoid\ (W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = sigmoid\ (W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = sigmoid\ (W_o x_t + U_o h_{t-1} + b_o)$$

$$u_t = sigmoid\ (W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot u_t$$

$$h_t = o_t \odot \tanh(c_t)$$

5. **GRU:** Gated Recurrent units are a simplified extension of LSTM, while its role remains the same. It has two-fold difference – gates (number of which are 2) and weights. It has no control over memory content owing to the absence of output gate. The previous activation's data flow is controlled by the update gate (1) as well as the new information (2), whereas a reset gate (3) is attached to said activation.

$$z_t = sigmoid(\ W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = sigmoid(\ W_r x_t + U_r h_{t-1} + b_r)$$

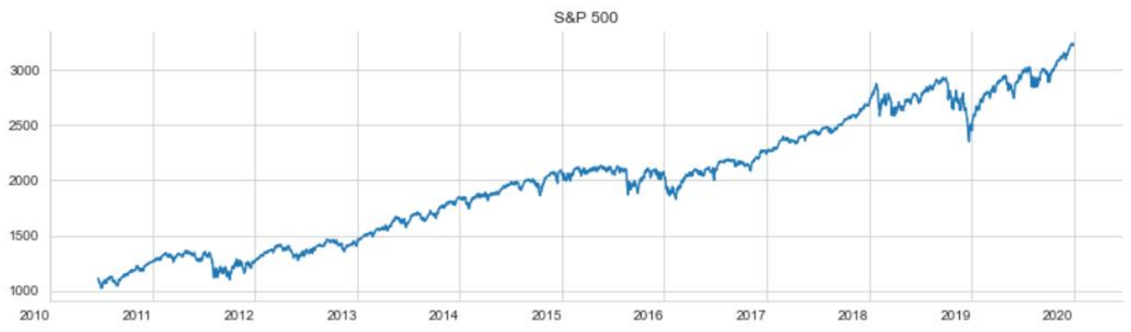
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh(\ W_h x_t + U_h(r_t \odot h_{t-1}\ ) + b_h$$

3. PROCEDURE

(A) Univariate Time-Series Regression to predict S&P500

3.1 Data Preparation and Feature Engineering:

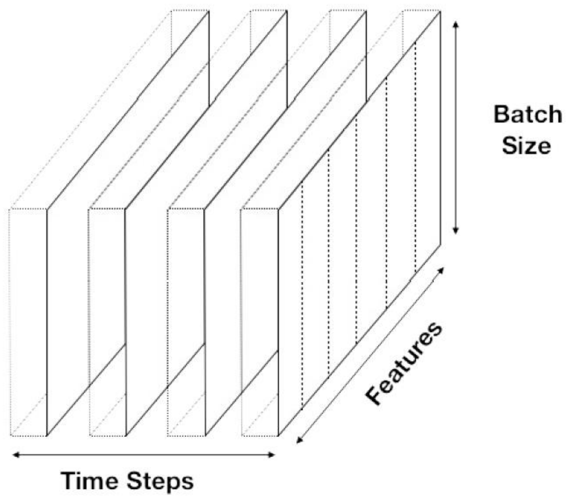
We first obtain the S&P 500 index values from 2010 to 2019 from the website of FRED using pandas data reader that directly gives us data in tabular format. The data looks as follows:



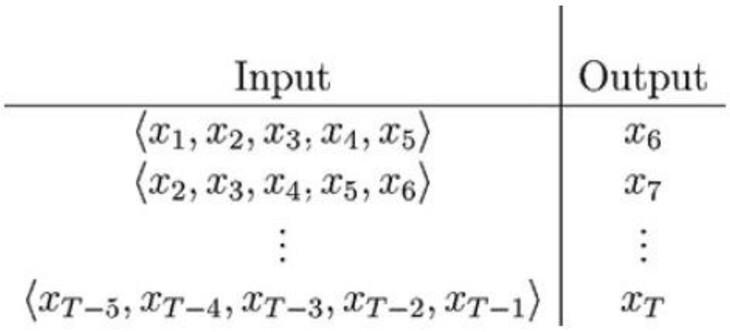
Smart feature engineering – that extracts useful features from unstructured data and prepares it for training – is an important step in training process. We use the $MinMaxScaler()$ to scale the data between $[0,1]$. The scaling process is given as follows:

$$\mathcal{X} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

We will simulate approx. 3 months or series of 63 consecutive trading days using one single LSTM layer that contains twenty hidden units to forecast one step ahead in time. Each LSTM layer must have 3 dimensions namely Batch Size, Features, Time Steps represented as:



For our data, S&P 500 has 2,463 time steps (observations). For a classification task we need a sliding window of 63 observations that will be utilized to create overlapping sequences from the rescaled data. A sample window using first five lags is shown as:



Input	Output
$\langle x_1, x_2, x_3, x_4, x_5 \rangle$	x_6
$\langle x_2, x_3, x_4, x_5, x_6 \rangle$	x_7
$\vdots$	$\vdots$
$\langle x_{T-5}, x_{T-4}, x_{T-3}, x_{T-2}, x_{T-1} \rangle$	x_T

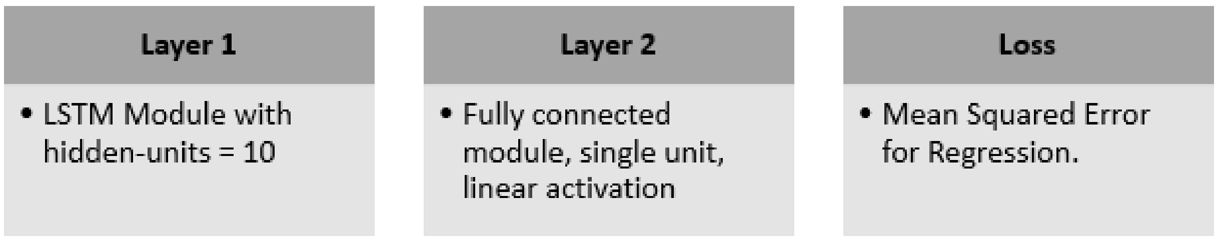
In generalized format for a “S” sized window we have:

$$x_t = f(x_{t-1}, x_{t-2}, \dots \dots, x_{t-s}) \ \forall \, t = S, S+1, \dots T$$

The 2019 data will be used as test data after reshaping features to add necessary third dimension. This is called as the train-test split.

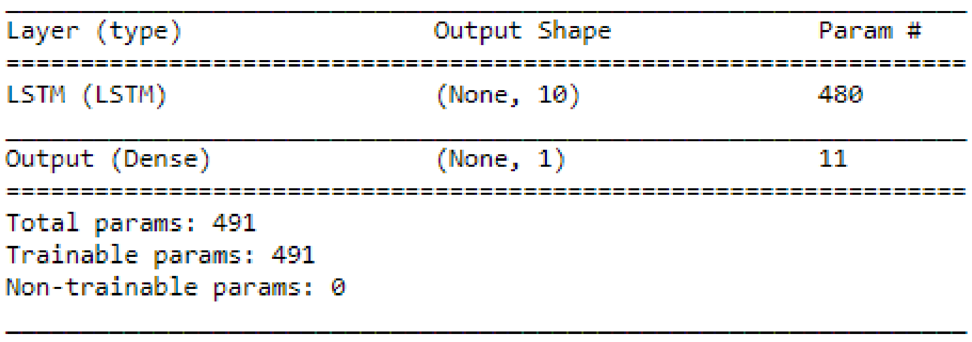
3.2 Model Architecture:

$Keras$ from $TensorFlow$ has all the necessary inbuilt functionality to build our requisite, single LSTM and two-layer RNN. The specifications for model are:



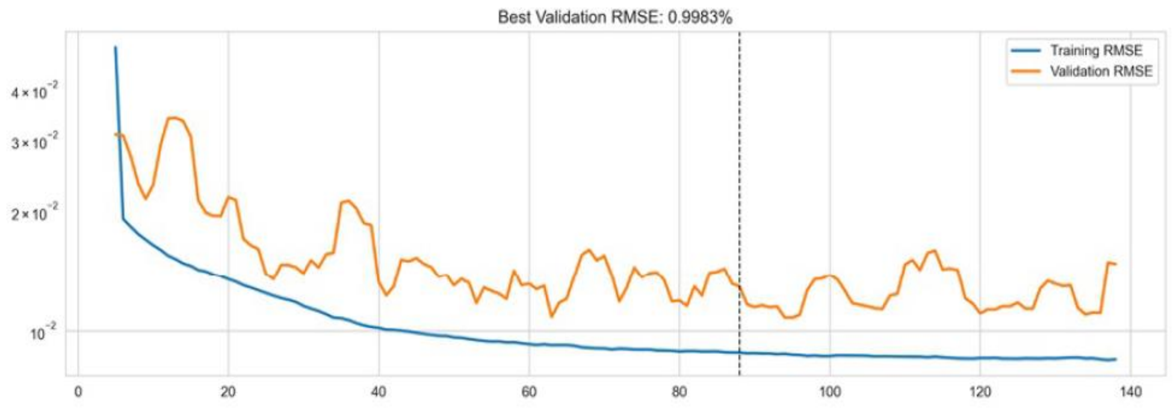
Layer 1	Layer 2	Loss
• LSTM Module with hidden-units = 10	• Fully connected module, single unit, linear activation	• Mean Squared Error for Regression.

Few important arguments for LSTM class are the ***Activation Function, Optimizer*** (we have used $RMSProp$ optimizer with default in built settings), ***Loss*** (as mean-squared-error for a regression-type problem), with an ***Early Stopping*** callback to train for 500 ***epochs***. The model summary shows a total of 491 parameters.

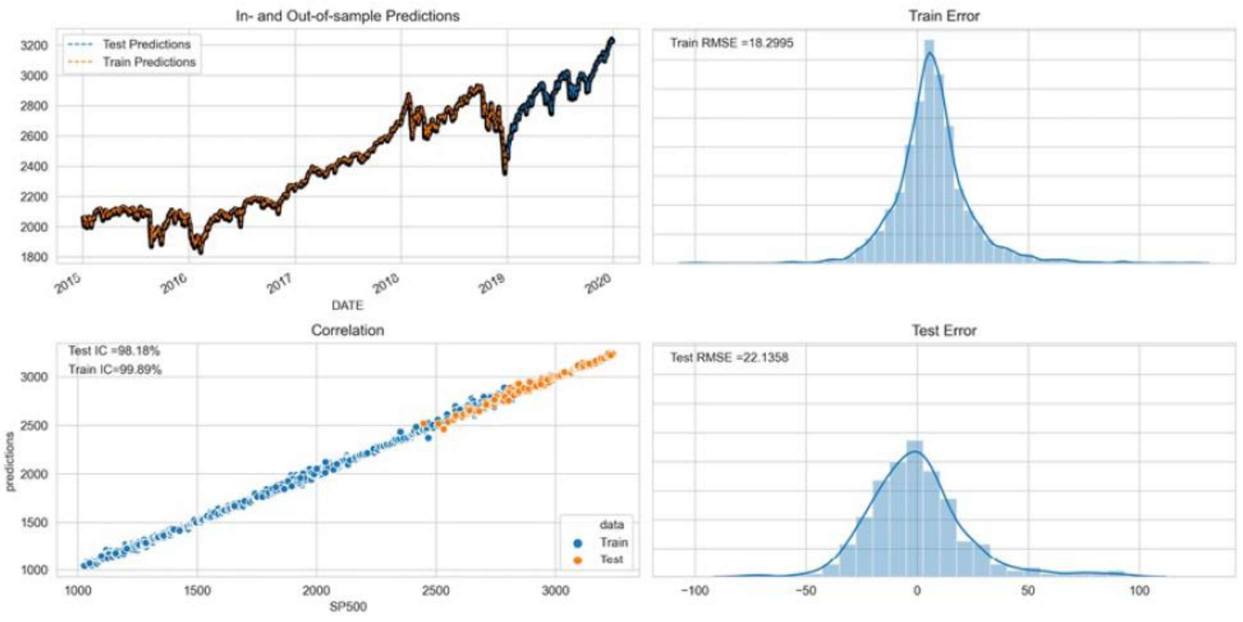


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
LSTM (LSTM)                  (None, 10)                480

_________________________________________________________________
Output (Dense)               (None, 1)                 11
=================================================================
Total params: 491
Trainable params: 491
Non-trainable params: 0

_________________________________________________________________
```

After 139 epochs, the training stops. The below figure shows a five epoch rolling average of validation and training Root Mean Square Error, signifying the best epoch with a loss of 0.998%.

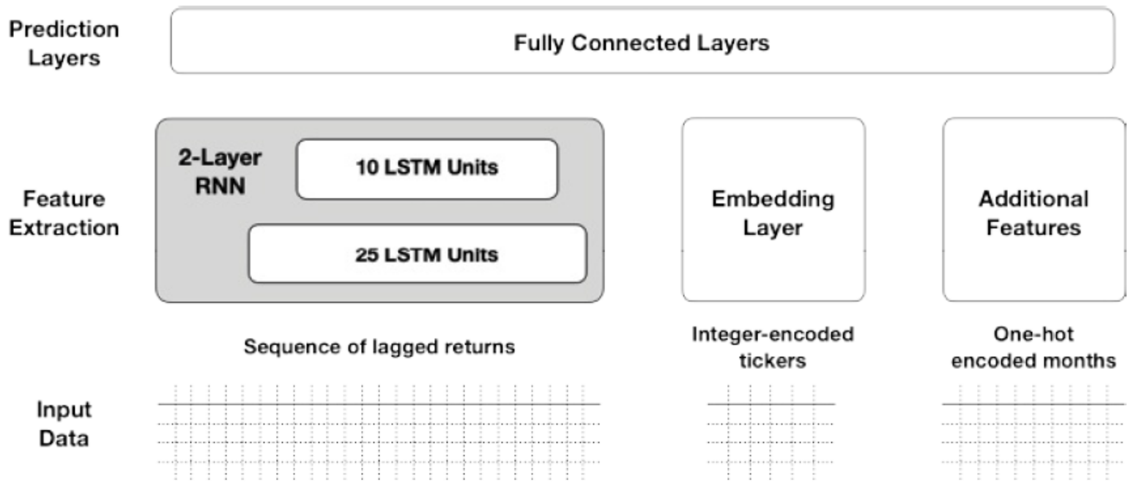


The out-of-sample performance – i.e. on the rescaled 2019 data with IC of 0.9899 – is given below.



(B) Stacked LSTM – predicting price moves and returns

A stacked and deeper model is now built using two LSTM layers using stock price data from *Quandl*. Non-sequential features like indicator variables that assist in describing the type of equity and month will also be incorporated into the mix. A sample of stacked LSTM with multiple features is given as:

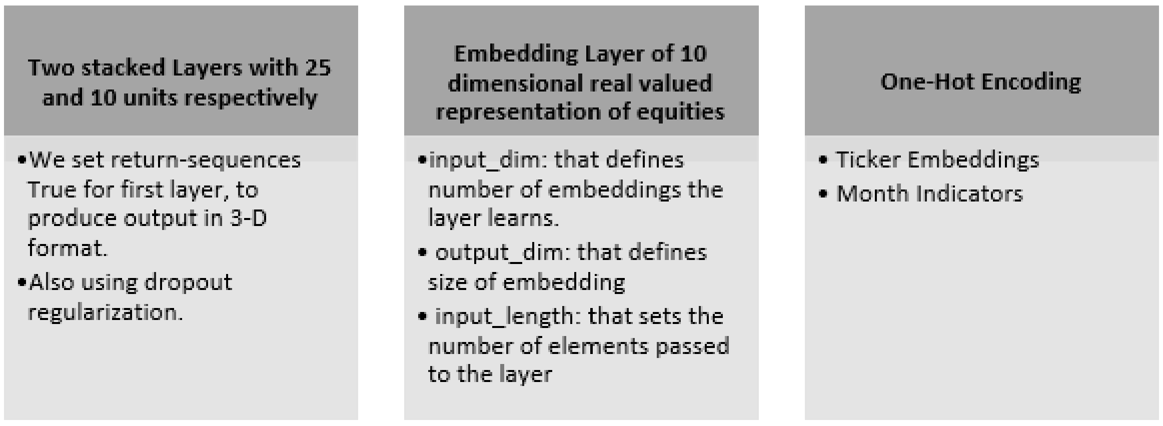


3.3 Data Preparation and Feature Engineering:

The weekly returns data is gathered for ~2500 stocks in time-period 2008-2017. A stack of rolling sequences of 52 weekly returns for each ticker and week is generated followed by removing outliers to data at 1 & 99 percent level, along with a label indicating positive or negative weekly return. Multiple inputs comprise of (1) 52 weeks of lagged returns (2) encoding for each of 12 months (3) encoding for tickers. The train data comprises of 2009-2016 data, while a hold out set of 2017 data is created.

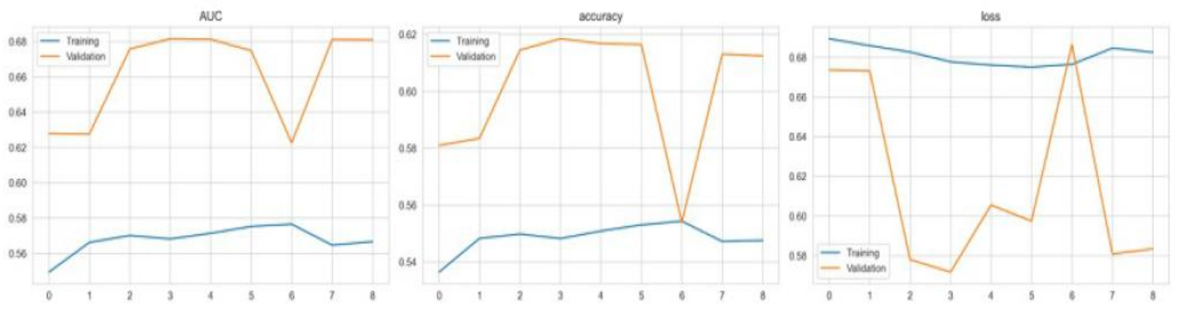
3.4 Model Architecture:

The model architecture is defined as follows:



Two stacked Layers with 25 and 10 units respectively	Embedding Layer of 10 dimensional real valued representation of equities	One-Hot Encoding
•We set return-sequences True for first layer, to produce output in 3-D format. •Also using dropout regularization.	•input_dim: that defines number of embeddings the layer learns. • output_dim: that defines size of embedding • input_length: that sets the number of elements passed to the layer	• Ticker Embeddings • Month Indicators

We shall use the RMSProp optimizer with default settings and compute the AUC metric that we'll use for early stopping (50 epochs) on the training set. The below plots shows the cross validation perfomance on **Stacked LSTM classification**, the training stopping after 8 epochs and evaluate the model performance.



Test AUC	**Test Accuracy**	**IC for test predicition and actual weekly returns**
0.685607154498662	0.62186058141098694	0.3175589786542647

To predict numerical returns rather than simply directional movements (i.e. a regression task), we do some minor changes as follows.

Select the *fwd_returns* outcome instead of *label*

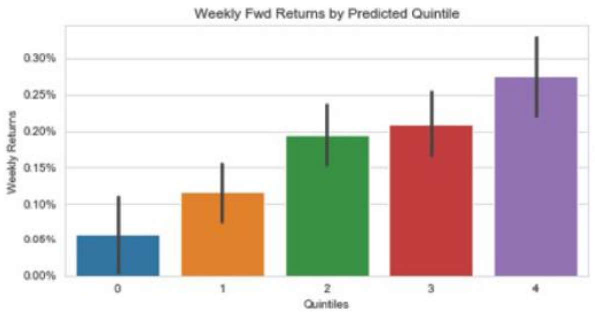
Convert the model output to *sigmoid* instead of linear (default)

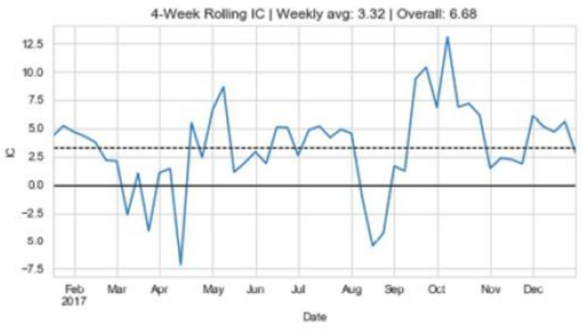
Update the loss to *mean squared error*

Updating optional metrics for regression tasks

The results of **Stacked LSTM regression** on out of sample data are:

Average Weekly IC	Entire Period IC	Return differential between top and bottom quintiles
3.32	6.68	20 basis points





4. CONCLUSION AND NEXT STEPS:

This paper aimed to explore and describe several concepts of the applying recurrent neural networks to time series forecasting (univariate), though it is by far not comprehensive. RNN's – tailored to sequential data – surely have an upper hand with respect to linear models used in time series forecasting. The paper began by reviewing a working architecture of RNN, analysis of the computational graph and RNN's overcoming of FFNNS and CNNs to capture long-range dependencies in data. We also reviewed Vanishing and Exploding gradient problems and saw how LSTM and GRUs assist in learning several time steps back. Finally, we applied a RNN in an algorithmic trading enviroment to predict univarite timeseries returns.

Neural networks don't provide much information in describing the process that drives time series, however they are certainy helpuful in forecasting processes. Mutli-period forecasts do generate higher errors and step ahead forecast. Direction change predicitions generated smaller errors too. Forecasting framework can be enhanced in several ways. RNN's can be improved by bagging. Feature engineering to improve input parameters can certainly help. For example, taking different features across different time periods to generate high quality forecasts. Lastly the choice of keras input parameters – like the choice of optimizers (Adam, SGD etc.) or hyperparameter tuning will certainly impact model performance. We did not aim to find best parameters for this paper using Grid Search or Random Search. There is so much left to be done. RNNs clearly deserve a seat in the toolbox of time series forecasting.

5. REFERENCES

- Sequence Modeling: Recurrent and Recursive Nets, Deep Learning Book, Chapter 10, Ian Goodfellow, Yoshua Bengio and Aaron Courville, MIT Press, 2016
- Understanding LSTM Networks, Christopher Olah, 2015
- An Empirical Exploration of Recurrent Network Architectures, Rafal Jozefowicz, Ilya Sutskever, et al, 2015

- S. Selvin , R. Vinayakumar , E. A. Gopalakrishnan , V. K. Menon and K. P. Soman.(2017) ”Stock price prediction using LSTM, RNN and CNN-sliding window model.” International Conference on Advances in Computing, Communications and Informatics: 1643-1647.
- Rather A. M., Agarwal A., and Sastry V. N. (2015). ”Recurrent neural network and a hybrid model for prediction of stock returns.”Expert Systems with Applications 42 (6): 3234-3241.
- Zhang G., Patuwo B. E., and Hu M. Y. (1998). ”Forecasting with artificial neural networks: The state of the art.” International journal of forecasting 14 (1): 35-62.
- Jabin S. (2014). ”Stock market prediction using feed-forward artificial neural network”. growth 99 (9).
- Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735-1780, 1997.
- Moghaddam A. H., Moghaddam M. H., and Esfandyari M. (2016) ”Stock market index prediction using artificial neural network.” Journal of Economics, Finance and Administrative Science 21 (41): 89-93.
- Roman J., and Jameel A. (1996). ”Backpropagation and recurrent neural networks in financial analysis of multiple stock market returns.” In Twenty-Ninth Hawaii International Conference on system sciences 2: 454-460.
- Mizuno H., Kosaka M., Yajima H. and Komoda N. (1998). ”Application of neural network to technical analysis of stock market prediction.”Studies in Informatic and control 7 (3): 111-120.
- Budhani N., Jha C. K., and Budhani S. K. (2014).” Prediction of stock market using artificial neural network.”In Soft Computing Techniques for Engineering and Technology (ICSCTET) : 1-8
- Zaiyong Tang, Chrys De Almeida, and Paul A Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. Simulation, 57(5):303-310, 1991.
- Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks: The state of the art. International journal of forecasting,14(1):35-62, 1998
- John Cristian Borges Gamboa. Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887, 2017*.
- Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555, 2014*.
- B. Bakker. Reinforcement Learning with Long Short-Term Memory. In Advances in Neural Information Processing Systems, 14, 2002.
- Wang J. Z., Wang J. J., Zhang Z. G. and Guo S. P. (2011). ”Forecasting stock indices with back propagation neural network.” Expert Systems with Applications 38 (11): 14346-14355.
- Karpathy A., Johnson J. and Fei-Fei L. (2015). ”Visualizing and understanding recurrent networks.”arXiv preprint arXiv :1506.02078
- Guresen E., Kayakutlu G., and Daim T. U. (2011). ” Using artificial neural network models in stock market index prediction.” Expert Systems with Applications 38 (8): 10389-10397
- Abinaya P., Kumar V.S., Balasubramanian P. and Menon V.K. (2016). ”Measuring stock price and trading volume causality among Nifty50 stocks: The Toda Yamamoto method.” In Advances in Computing, Communications and Informatics (ICACCI) :1886-1890