Applications of Machine Learning in Natural Language Processing and Time Series Forecasting

Siddharth Agrawal, Shaan Sheth, Shrey Sheth, Aarya Shah

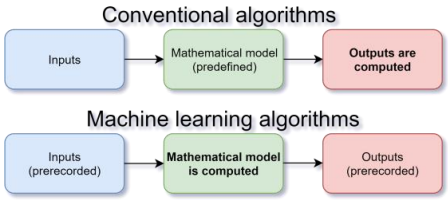
Abstract

Over the years, machine learning methods have developed to generate state-of-the-art models in several fields, with Natural Language Processing and Time Series Forecasting being some of the most prominent applications. An account of the history, structure, mathematics, advantages, and limitations of some machine learning techniques such as Markov chains, word2vec, Recurrent Neural Networks, Long Short-Term Memory, seq2seq, autoencoders, and transformers are discussed in this paper, with reference to their applications in Time Series Forecasting of stock prices and website-traffic, and various Natural Language Processing tasks such as autoregression, summarisation, downstreaming, classification, topic modelling, plagiarism detection, abstract generation, keywords extraction, etc.

Index Terms—Machine Learning, Natural Language Processing, Time Series Forecasting, Neural Networks, Deep Learning

Introduction

Artificial neural networks are a type of machine learning algorithm. Machine learning is a branch of discreet mathematics and computer science. Typical algorithms give an output based on predefined mathematical model and programming, and any input(s). Machine learning algorithms[1] take pre-recorded inputs and outputs to generate the mathematical model which can then be used to estimate outputs based on other inputs.



Defining the steps or formulating an algorithm to take inputs as previous stock data and information to predict future stock prices is extremely difficult as they have a very complex relationship. Neural networks are extremely versatile at determining these complex relationships between inputs to outputs.

In this investigation, we will be exploring several Machine Learning models such as SVM, and Markov Chain Model.

We will also be exploring several types of Neural Networks: Deep Artificial neural networks, recurrent neural networks, long-short term memory neural networks, Temporal Convolution Networks, and Transformers. Knowledge and understanding of each of the networks serves as a basis to understand the next network.

Mathematics of such machine learning models will be explored and then applied for the following tasks:

Using machine learning models and applying them on datasets such as: IMDB, Yelp-2, Yelp-5, Amazon-2, Amazon-5, 20newsgroups

Using machine learning models and applying them on several time series prediction tasks such as: electricity power consumption, web-traffic forecasting, stock-price prediction

Background

[1]Machine learning algorithms: Application of artificial intelligence in providing systems the ability to automatically generate, learn, and improve from experience by themselves, without explicit programming.

Time Series Forecasting includes various applications where the future data needs to be predicted. This includes climate and weather forecasting models, stock price prediction models, web-traffic prediction models, machine failure prediction models, etc. Time Series Forecasting is very varied and different models are useful in different applications. However, many of the models used overlap with models used in NLP therefore allowing us to explore more applications without the need of coding additional models.

There are many notations used within the report that are used in computer science, especially, artificial neural networks' nomenclature which may seem to be plagiarised but are, in fact, the works of the author.

Natural Language Processing is the application of computational techniques for natural language, speech, and text. It is used for processing, analysing, classifying, summarising, topic modelling, plagiarism detection, plagiarism rewriting, grammar checking, spell checking, etc. of text.

Most important aspects of an algorithm are its efficacy and efficiency. The predictive capabilities of the model (efficacy) have to be maximized, but the amount of time it takes to optimize the model (efficiency) has to be minimized. In order to increase efficiency, the no. of calculations, and the complexity of the calculations, calculated by the computer has to be reduced.

Relevant Machine Learning models include SVM, Markov Chain Model, ANN, RNN, LSTM, TCN, Transformers.

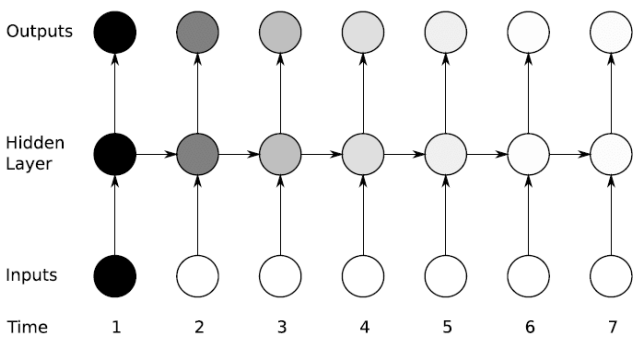
Motivation

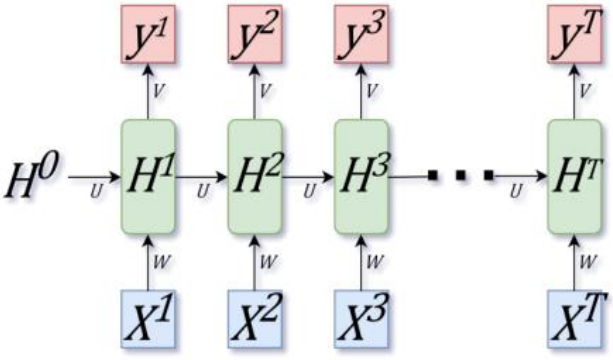
While plenty of literature and pretrained models as well as architectures are available in abundance. The in-depth mathematical and computational understanding of models used in NLP and TSF is lacking. This paper will be demystifying these models by delving deep into the mathematics and computations behind the ML models used for such applications. The paper will cover the basic ML implementations such as Markov Chains and networks such as Feed Forward Networks and then move on to more complex models such as Recurrent Neural Networks, Long Short-term memory Neural Networks, Seq2Seq networks, and Transformers.

Literature Survey

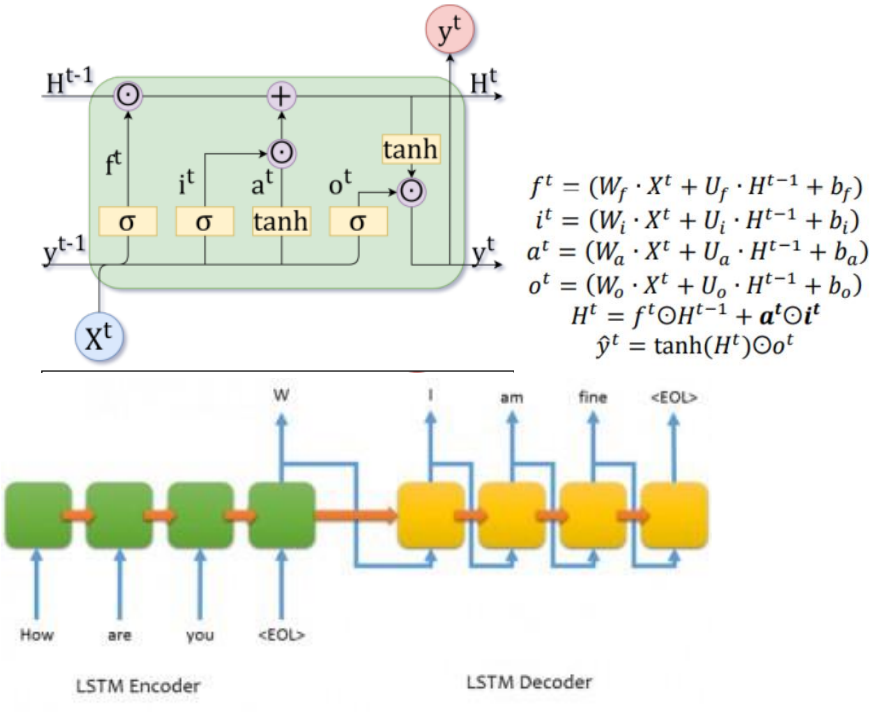
Conventional strategies of stock prediction such as news, buy and hold, martingales, momentum, mean reversion, etc. predict stocks with limited efficacy. With improvements in technology, stock trading has been transferred to automated computer predictions using ML.

RNN are the most basic form of sequence based neural network but it suffers from vanishing gradients.





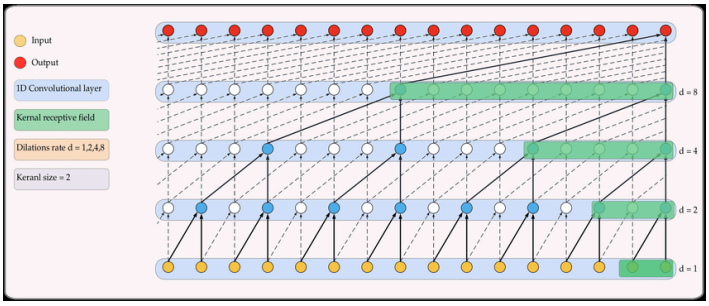
To address vanishing gradients problem, many NLP models used LSTM networks. Such networks had longer memory than RNN networks. Furthermore, they could be used on large documents and text sequences due to their recursive nature as the memory requirement did not depend on input sequence length.



$$f^t = (W_f \cdot X^t + U_f \cdot H^{t-1} + b_f)$$
$$i^t = (W_i \cdot X^t + U_i \cdot H^{t-1} + b_i)$$
$$a^t = (W_a \cdot X^t + U_a \cdot H^{t-1} + b_a)$$
$$o^t = (W_o \cdot X^t + U_o \cdot H^{t-1} + b_o)$$
$$H^t = f^t \odot H^{t-1} + \boldsymbol{a^t} \odot \boldsymbol{i^t}$$
$$\hat{y}^t = \tanh(H^t) \odot o^t$$

Later seq2seq models developed by google which produced higher accuracy and efficacy in various NLP and time series tasks. It used an encoder and a decoder model. One of the LSTMs would encode the text into another language and the other would try to decode it. Though it was first designed for translation and text2speech or speech2text tasks, it outperformed conventional LSTMs in other NLP tasks as well. This was because this architecture facilitated unsupervised learning.

New architectures that used convolution on time series were also effective. Called Temporal Convolutional Networks (TCN), they also outperformed LSTM in tasks that required very long sequences. The text or input sequence is divided into smaller sequences, which are then pooled together to get a new set of sequences that accounted for larger percentage of the overall input sequence. This process is repeated till the input sequence is sufficiently small and then mapped to outputs after through a dense layer.



They addressed vanishing gradient problem that still exists in LSTM (though to a much lesser degree) by using an approach similar to divide-and-conquer.

They exhibit longer memory than recurrent architectures like RNN, LSTM, GRU and are able to perform better on vast range of tasks such as Word-level PTB, MNIST, Adding Problem, Copy Memory, etc.

They also allowed for parallelism which recurrent architectures do not (as they need the output from the first input/timestep in order to feed the data into itself recursively and produce outputs for future timesteps).

2017 was a great year for NLP progress. With papers such as “Efficient Estimation of Word Representations in Vector Space” and “Attention Is All You Need” being published. “Efficient Estimation of Word Representations in Vector Space” covered a new architecture called Transformers that could be used to encode vector representations of words. This technique, commonly known as word2vec in NLP nomenclature became very popular and effective as it could represent large vocabulary as a vector space which reduced the dimensionality and memory issues with one-hot encoding. “Attention Is All You Need” served as a landmark paper as it changed our conventional understanding of NLP models away from TCN and LSTM networks onto Transformer and self-attention based networks.

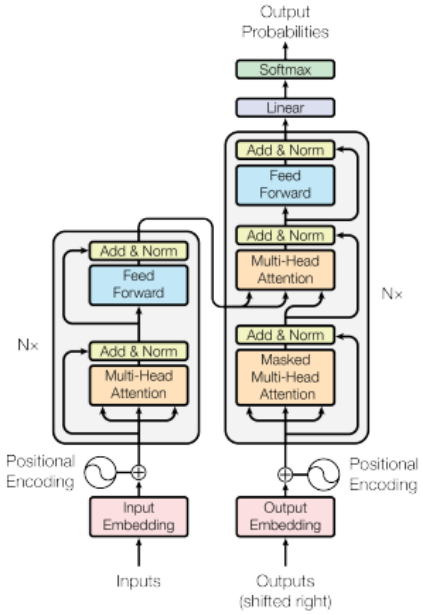


Figure 1: The Transformer - model architecture.

Transformer based self-attention networks have several advantages over older network architectures.

- Reduced worst case complexity for path length thus reducing vanishing gradient problem.
- Increased parallelisation capabilities due to its non-recursive nature.
- Bi-directionality: words can be placed in their context as the model has access to the words both before and after the token/word to be predicted. Therefore, understanding textual context better and increasing accuracy in benchmarks.
- Is used in all current generation models including GPT2, XL-Nets, BERT, RoBERTa, GPT3, etc.
- Disadvantage: Cannot be used for long text sequences as the space and computational complexity is O(n^2). Though a potential remedy is sparse attention networks.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Section I Prior knowledge required

1.1 Hadamard Product or element-wise product (symbol ⊙)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,J} \\ \vdots & \ddots & \vdots \\ x_{I,1} & \cdots & x_{I,J} \end{bmatrix} \odot \begin{bmatrix} y_{1,1} & \cdots & y_{1,J} \\ \vdots & \ddots & \vdots \\ y_{I,1} & \cdots & y_{I,J} \end{bmatrix} = \begin{bmatrix} x_{1,1} \times y_{1,1} & \cdots & x_{1,J} \times y_{1,J} \\ \vdots & \ddots & \vdots \\ x_{I,1} \times y_{I,1} & \cdots & x_{I,J} \times y_{I,J} \end{bmatrix}$$

Where mat(x) has I rows and J columns, and, mat(y) is of same size. Note that matrix hadamard product is only defined if size of the matrices are equal.

This can be generalised for all elements $i < I,\ i \in N\ and\ j < J, j \in N$ as:

$$z_{i,j} = x_{i,j} \times y_{i,j} \quad where, mat(z) = mat(x) \odot mat(y)$$

1.2 Dot Product or Inner Matrix Product (symbol ·)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,J} \\ \vdots & \ddots & \vdots \\ x_{I,1} & \cdots & x_{I,J} \end{bmatrix} \cdot \begin{bmatrix} y_{1,1} & \cdots & y_{1,K} \\ \vdots & \ddots & \vdots \\ y_{J,1} & \cdots & y_{J,K} \end{bmatrix} = \begin{bmatrix} \sum_{n=1}^{J} x_{1,n} \times y_{n,1} & \cdots & \sum_{n=1}^{J} x_{1,n} \times y_{n,K} \\ \vdots & \ddots & \vdots \\ \sum_{n=1}^{J} x_{I,n} \times y_{n,1} & \cdots & \sum_{n=1}^{J} x_{I,n} \times y_{n,K} \end{bmatrix}$$

Where mat(x) has I rows and J columns, and, mat(y) has J rows and K columns. Note that matrix dot product is only defined if number of columns of the first matrix = the number of rows of the second matrix.

This can be generalised for all elements $i < I,\ i \in N\ and\ j < J,\ j \in N\ and\ k < K, k \in N$ as:

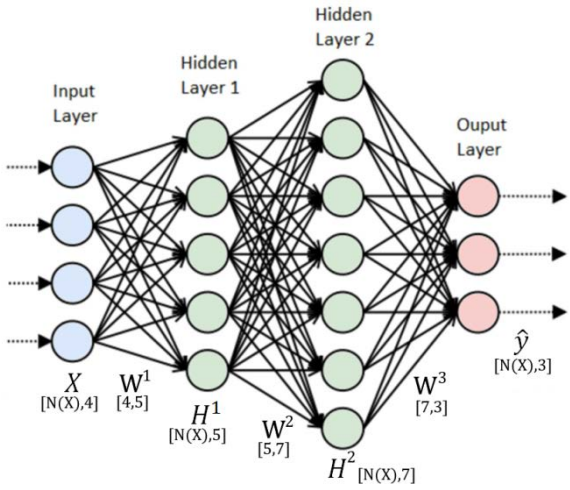
$$z_{i,j} = \sum_{n=1}^{J} x_{i,n} \times y_{n,j} \quad where, mat(z) = mat(x) \cdot mat(y)$$

Section II Deep Artificial Neural Networks

Neural networks have 3 major steps: 1st forward propagation, 2nd back-propagation, and 3rd update weights. These are repeated several times, leading to more and more optimized model with each iteration.

2.1: Understanding Deep Artificial neural networks (DANN)

2.1.1 Forward propagation



Matrix X: input data (represented by the input layer). Inputs contain current stock price data and/or other factors which may affect future stock prices.

Matrices W: weights (represented by the arrows). Each arrow may represent a different magnitude. Random variables for weights are generated for the first iteration.

Matrices H: (represented by Hidden layers) calculated by the following operations:

$$\left.\begin{aligned} \sigma(X \cdot W^1) &= H^1 \\ \sigma(H^1 \cdot W^2) &= H^2 \\ &\vdots \\ \sigma(H^{N-1} \cdot W^N) &= H^N \end{aligned}\right\} N-1\ times$$
$$\sigma(H^N \cdot W^{N+1}) = \hat{y}$$

Here, $\sigma(x) = \frac{1}{1+e^{-x}}$ also known as the 'sigmoid function[6]' and this function is applied element-wise. N denotes the number of hidden matrices. Note: Matrix and vector dimensions need to be such that they have a defined dot product.

Vector $\hat{y}$: predicted output, in this case, future stock data (represented by the output layer).

The aim of neural networks is to generate a set of matrices W, such that, when the aforementioned set of operations are applied on vector X (inputs), it should result in vector $\hat{y}$ (predicted output) that is ≡ vector y (actual outputs i.e. actual stock price). These weights are optimized by Back-Propagation.

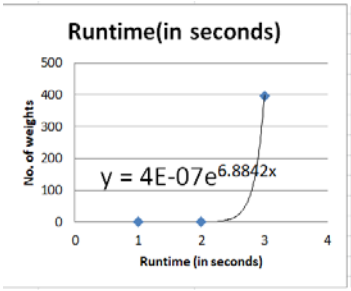
2.1.2 Back propagation

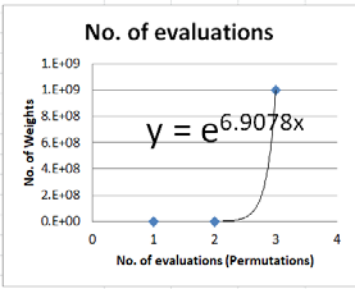
Our predictions from this forward-propagation are probably extremely incorrect as a random weight matrix was generated. To improve this model, the incorrectness of the predictions must be quantified. This is done with the total error-squared function $E = \sum \frac{1}{2}(y - \hat{y})^2$ (summation of the difference between all predicted outputs and all real outputs, multiplied by $\frac{1}{2}$ and squared).

[6] Please refer to 2.1.2 back propagation(next page) for the justification of utilizing sigmoid function.

The objective of the neural network is to minimize this total error-squared. Total error-squared is minimized by optimizing weights (as inputs and outputs cannot be changed as they are real stock prices). A brute force approach may be used: adjust all weights until cost is minimized. However, this poses the problem known as 'curse of dimensionality'. Best explained by the following example:

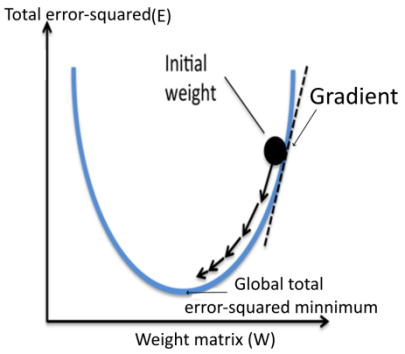
Let's assume that only 1 weight value has to be optimized, and 1000 different weight values are tested. Using my computer, and a hypothetical dataset, this took 0.0004162 seconds. But as the number of weights increase, in order to maintain the same precision (1000), the runtimes increase exponentially as below:



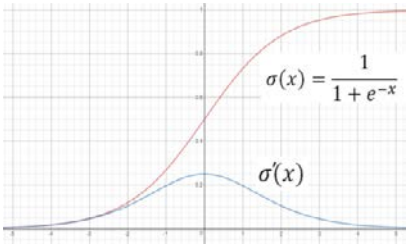


This means that the complexity[7] of brute force algorithm is 1000^n where n is the number of weights. This means that for optimization of 10 weights would take approximately an astounding 3.1605358E+23 seconds or $= (4 \times 10^{-7})e^{6.8842(10)}$(Taken from figure to the left) (Time period longer than the universe's existence).

This is incredibly inefficient; however, a better approach can be taken. During calculus classes we have learnt optimization. If relationship between weights and total error-squared was known, then finding the minimum point by equating $\frac{dE}{dW} = 0$ would optimize the weights (minimize total error-squared). However, it is not that simple. The cost function is dependent on weights which are dependent on inputs; and a clear relationship between weights, inputs, and outputs is not currently known. So, another approach has to be taken: the direction which is downwards can be taken into consideration. Knowing the rate at which the total error-squared changes in relation to change in the weights$\left(\frac{\partial E}{\partial W}\right)$, the weights are updated to move in the negative direction of this slope. This is repeated multiple times, moving closer to the global minimum point with each iteration, as total error-squared function is similar to quadratic in nature with only 1 minima (rightward diagram (simplified)). This method is known as gradient descent.



Note: partial derivative of total error-squared is taken with respect to each weight matrix as one weight matrix is taken into account at a time and other relationships are ignored.



The reason why activation function[8] like sigmoid is used is because of their non-linear property which makes them differentiable, and thus optimizable. **Sigmoid Activation Function** is used for neural networks as it mimics normal probability distributions (if positive x values are reflected) and probability exists only between the range of **0 and 1**. It also mimics brain's action potential as when x (similar to electrochemical potential) is negative, the neuron does not let information pass through, but, as x increases the neuron lets information pass through.

The cost function is dependent on output $\hat{y}$, which is further dependent on $H^N \; and \; W^{N+1}$, which are further dependent on $H^{N-1} \; and \; W^N$ and so on until H^1 is dependent on $X \; and \; W^1$.

Notation: N denotes the number of hidden matrices, $\infty > N \geq n > 1 \; and \;\; N, n \in Natural \; numbers$

$$\frac{\partial E}{\partial W^n} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial H^N} \overbrace{\frac{\partial H^N}{\partial H^{N-1}} \cdots \frac{\partial H^{n+1}}{\partial H^n}}^{N-n \; times} \frac{\partial H^n}{\partial W^n} \tag{1}$$

$$= \overbrace{\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial (H^N \cdot W^{N+1})}}^{[1]} \overbrace{\frac{\partial (H^N \cdot W^{N+1})}{\partial H^N}}^{[2]} \overbrace{\frac{\partial H^N}{\partial (H^{N-1} \cdot W^N)} \frac{\partial (H^{N-1} \cdot W^N)}{\partial H^{N-1}}}^{[2]} \cdots \overbrace{\frac{\partial (H^{n-1} \cdot W^n)}{\partial W^n}}^{[3]} \tag{2}$$

$$\frac{\partial E}{\partial W^{N+1}} = [1][2] \cdots \overbrace{\frac{\partial (H^N \cdot W^{N+1})}{\partial W^N}}^{[3]} \qquad \frac{\partial E}{\partial W^1} = [1][2] \cdots \overbrace{\frac{\sigma(X \cdot W^1)}{\partial W^1}}^{[3]} \tag{2}$$

[7] Activation function is usually used to map the outputs between desired range (0 to 1) or (-1 to 1).
[8] Complexity is a relationship that describes runtime of an algorithm.

using quotient rule,

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{\cancel{(1+e^{-x})(0)} - (1)(-e^{-x})}{(1+e^{-x})^2}$$

$$= \left(\frac{1}{(1+e^{-x})}\right)\left(\frac{-e^{-x}}{(1+e^{-x})}\right)$$

$$= \left(\frac{1}{(1+e^{-x})}\right)\left(\frac{1+e^{-x}-1}{(1+e^{-x})}\right)$$

$$= \left(\frac{1}{(1+e^{-x})}\right)\left(\left(\frac{\cancel{1+e^{-x}}}{\cancel{(1+e^{-x})}}\right) - \left(\frac{1}{(1+e^{-x})}\right)\right)$$

$$= \sigma(x)(1-\sigma(x)) \quad (3)$$

Solving [1] by using power rule and substituting (3)

$$\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial (H^N \cdot W^{N+1})} = \sum_{m=1}^{M} (y_m - \hat{y}_m) \odot \left(\sigma(\hat{y}_m) \odot \left(1 - \sigma(\hat{y}_m)\right)\right) = \delta^N \quad (4)$$

In back propagation of neural networks, some matrices may be transposed[9], and, the order of calculating matrix products may also be changed. This is done so that the matrix dimensions are such that the matrices line up properly and can have a defined dot product. Furthermore, doing so has the advantage of adding up all the errors through the process of dot product allowing for the computation of $\frac{\partial E}{\partial W^n}$ as in example on right: Where, K is the no. of columns and M is the no. of rows of the particular hidden matrix.

$$\begin{bmatrix} \sum_{m=1}^{M} H_{m,1}^N \cdot \delta_m^{N+1} \\ \sum_{m=1}^{M} H_{m,2}^N \cdot \delta_m^{N+1} \\ \vdots \\ \sum_{m=1}^{M} H_{m,K}^N \cdot \delta_m^{N+1} \end{bmatrix} = (H^N)^{Tp} \cdot \delta^{N+1} = \frac{\partial E}{\partial W^N} \quad (5)$$

Notation: $\left(mat(x)\right)^{\boldsymbol{Tp}}$ denotes transposed matrix of any matrix x.

Solving [2] by substituting(3). similar to (5)

$$\frac{\partial (H^n \cdot W^{n+1})}{\partial H^n} \frac{\partial H^n}{\partial (H^{n-1} \cdot W^n)} = (W^{n+1})^{Tp} \odot \left(\sigma(H^n) \odot \left(1 - \sigma(H^n)\right)\right) = \delta^n \quad (6)$$

Solving [3]

$$\frac{\partial (H^{N-1} \cdot W^N)}{\partial W^N} = H^N \qquad \frac{\partial (H^{n-1} \cdot W^n)}{\partial W^n} = H^{n-1} \qquad \frac{\sigma(X \cdot W^1)}{\partial W^1} = X \quad (7)$$

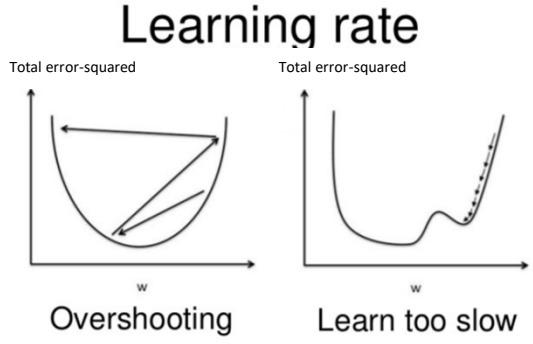
Substituting [1], [2] and [3] into (2), similar to (5), for final derivative as follows:

$$\frac{\partial E}{\partial W^N} = (H^N)^{Tp} \cdot \delta^{N+1} \qquad \frac{\partial E}{\partial W^n} = (H^n)^{Tp} \cdot \overbrace{\delta^{N+1} \cdot \delta^N \cdots \delta^n}^{n+1\ times} \qquad \frac{\partial E}{\partial W^1} = (X)^{Tp} \cdot \overbrace{\delta^N \cdot \delta^{N-1} \cdots \delta^1}^{N+1\ times} \quad (8)$$

2.1.3 Update weights:

All new weights are computed by: $W(new)^i = W(old)^i - \lambda \odot \frac{\partial E}{\partial W^n} W(old)^i$,

here $\lambda = learning\ rate\ and \infty > N \geq i \geq 1$



The learning rate is usually a value ≥0.3. This is a user-defined constant that defines the rate at which gradient descent occurs. If it is too high, model may overshoot the global minima, and if it is too low, model may take a lot of time and iterations to reach the global minimum point.[10]

This entire 3 step process is repeated several times, till the relationship between total error-squared and weights reaches convergence (minimized). The optimized weight matrices can then be utilised on new sets of input data to predict future stock prices by following the forward propagation calculations. These 3 steps remain largely same for all types of neural networks and therefore, will not be explained again in the next sections unless explicitly required.

[9] A transposed matrix is a matrix whose original rows and columns are switched to obtain a new matrix.

[10] "A Primer on how to optimize the Learning Rate of Deep Neural Networks", by Timo Böhm, *Towards Data Science*, https://towardsdatascience.com/learning-rate-a6e7b84f1658 Accessed on: 19 August, 2018.

2.2 Application of DANN:

First, all the data needs to be normalised by the min-max function: $minmax(x) = \frac{(x-min)}{(max-min)}$ where min is the lowest value in the dataset and max is the maximum value in the dataset. This scales all these values between 0 and 1. This helps the neural network relate between multiple factors well as they are now on the same scale. It also allows for faster movements during gradient descent as it is able to relate the input data with output data better as they are also now on the same scale. The model was tested against 2018 stock price data. These minmax normalised values are later inputted through the inverse minmax function: $minmax^{-1}(x) = x(max - min) + min$ to obtain the identity function (original values): $minmax(x)\left(minmax^{-1}(x)\right) = f(x) = x$.

Opening price	Normalized value
8210.10	0.250933
8196.05	0.247962
8202.65	0.249358

The stock data was optimized in mini-batches with data from 50 days at a time and input matrices of size 50×6 due to the following factors being considered at each day.

Date	Open price	High	Low	Volume	Moving average	Momentum
28-04-15	8215.55	8308	8185.15	174600	8374.41	-200.55
29-04-15	8274.8	8308.2	8219.2	160400	8350.86	-125.60

Moving average and momentum are calculated for each day as follows: (t denotes the day)

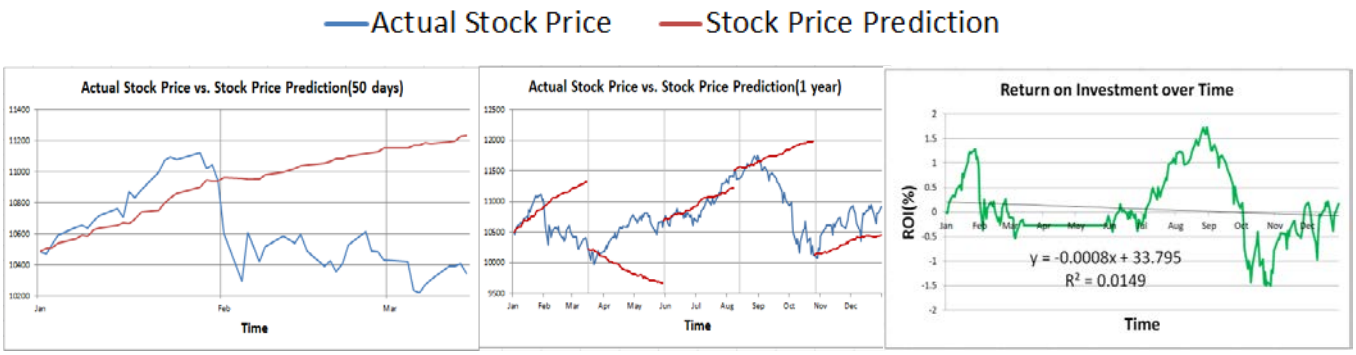
$$moving\ average^t = \frac{\sum_t^{t-5} opening\ price^t}{5} \qquad momentum^t = opening\ price^t - opening\ price^{t-5}$$

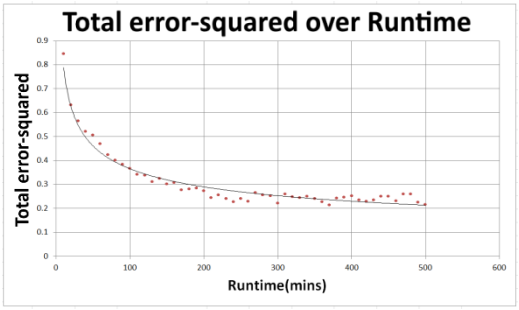
8 Weight matrices were used, all of size 50×50. Predicted output vector (future stock price prediction) is of size 50. Learning rate was set equal to 0.05. These optimized weights are then used to make predicted outputs(stock-price predictions) for the future.

2.3 Evaluation and Reflection

$$Standard\ deviation = \sqrt{\frac{\sum(x-\bar{x})^2}{n}} \qquad Return\ on\ interest\ ROI = \frac{Total\ returns}{Total\ invested} \times 100\%$$

Here, x is actual stock price data and $\bar{x}$ is predicted stock price data.





Standard deviation is high at 565.194. This is very inaccurate. Correlation coefficient(R) of the trendline and actual returns is low at 0.0149. This suggests that there is a constant return in investment. Fluctuations from this trendline are high, which suggests that investments based on DANN's predictions are high risk. The time to optimize the weights (regression time) took ≈450 minutes. Total error-squared reduced to ≈0.345.

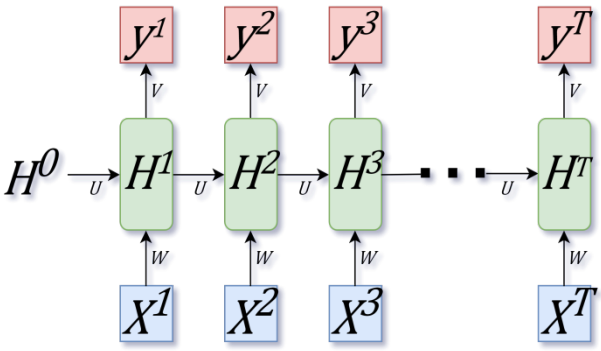
This model may be improved by accounting for temporal relationships as RNN and LSTM neural networks do.

Section III Recurrent Neural Networks (RNN)

RNNs are used when the sequential or temporal aspects of the data are more important than the spatial aspect of the data. This may be the case with stock price data as it is sequential in nature, wherein the past stock price data may affect the future stock prices.

3.1: Understanding Recurrent Neural Networks

3.1.1 Forward Propagation:



W: weight matrix, U: recurrent weight matrix, V: output weight matrix. X^t: input vector at time t. $\hat{y}^t, y^t$ are scalars, denoting predicted outputs, and actual outputs, respectively, at time t. H^0 and U^0 are null matrices[11]. U, V, and W matrices are filled with random numbers. U, and all H matrices are of same dimensions.

Then the following function is used **recursively**, to calculate H matrices.

$$\sigma(X^t \cdot W + H^{t-1} \cdot U) = H^t$$
$$\sigma(H^t \cdot V) = \hat{y}^n$$

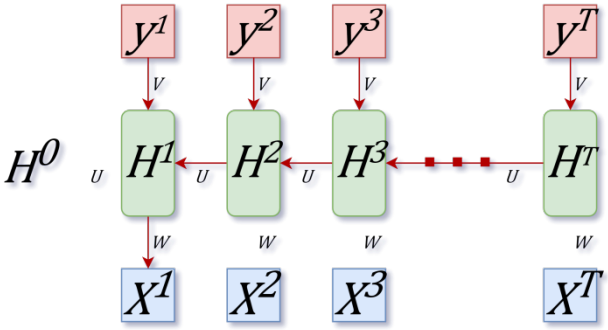
The major difference between DANN and RNN is that *Hidden matrix of* ***previous time*** $(H^{t-1}) \cdot$ *Recurrent weight matrix*(U) is **added** to the *input of* ***current time*** $(X^t) \cdot$ *weight vector*(W). This allows the past data to have an effect on the current hidden matrix, and consequently output vector, and subsequent hidden matrices. By doing this, both, the past data and the current data can be utilized to make future stock predictions.

In RNN, the **recurrent weight matrices** handle how the hidden matrix and input vectors of **previous time(s)** affect the **current time** hidden matrix, and consequently predicted output. The **weight matrices** handle how the input data of the **current time** affects the **current time** predicted output. Both weight matrices and recurrent weight matrices work in tandem to produce the predicted output.

3.1.2 Back propagation

Notation: T is the largest time-step in the LSTM sequence. $\infty > T \geq t > 0 \; and \; T, t \in N$

Notation: Total Error squared: $E = \sum_{t=1}^{T} \frac{1}{2}(y^t - \hat{y}^t)^2$ Error squared $\varepsilon^t = \frac{1}{2}(y^t - \hat{y}^t)^2$



$$\frac{\partial E}{\partial V} = \sum_{t=1}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial V}$$

As error ε^t is dependent on $H^{t-1} \cdots H^1$, while calculating differential $\frac{\partial \varepsilon^t}{\partial W}$, all hidden matrices $H^{t-1} \cdots H^1$ have to be differentiated using chain rule:

$$\frac{\partial \varepsilon^t}{\partial W} = \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^3}{\partial H^2} \frac{\partial H^2}{\partial H^1} \frac{\partial H^1}{\partial W}$$

And as the derivative of the summation of all errors squared is to be taken with respect to W, and U:

$$\frac{\partial E}{\partial W} = \frac{\partial \varepsilon^T}{\partial W} + \frac{\partial \varepsilon^{T-1}}{\partial W} + \cdots \frac{\partial \varepsilon^1}{\partial W}$$

$$\frac{\partial \varepsilon^t}{\partial W} = \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^3}{\partial H^2} \frac{\partial H^2}{\partial H^1} \frac{\partial H^1}{\partial W}$$

$$\therefore \frac{\partial \mathrm{E}}{\partial W} = \sum_{t=1}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \overbrace{\frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^2}{\partial H^1}}^{t-1 \; times} \frac{\partial H^1}{\partial W}$$

$$\frac{\partial E}{\partial U} = \frac{\partial \varepsilon^T}{\partial U} + \frac{\partial \varepsilon^{T-1}}{\partial U} + \cdots \frac{\partial e^1}{\partial U}$$

$$\frac{\partial \varepsilon^t}{\partial W} = \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^3}{\partial H^2} \frac{\partial H^2}{\partial U}$$

$$\frac{\partial \mathrm{E}}{\partial U} = \sum_{t=2}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \overbrace{\frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^3}{\partial H^2}}^{t-2 \; times} \frac{\partial H^2}{\partial U}$$

(9)

[11] Null matrix is a matrix whose all elements are equal to 0.

Notation: $\sigma'(mat(x)) = mat(x) \odot \big(1 - mat(x)\big)$ (using (3))

Unlike ANN most matrices are of equal sizes and therefore not much transposition is required, rather, element-wise products are calculated between matrices in back propagation with the exception of X which has to be transposed similar to ANN equation (5).

$\frac{\partial \varepsilon^t}{\partial \hat{y}^t} = \frac{\partial}{\partial \hat{y}}\left(\frac{1}{2}(y^t - \hat{y}^t)^2\right) = (y^t - \hat{y}^t)$	$\frac{\partial \hat{y}^t}{\partial H^t} = \sigma'(\hat{y}^t) \odot V$	$\frac{\partial H^t}{\partial H^{t-1}} = \sigma'(H^{t-1}) \odot U$	(10)
$\frac{\partial H^1}{\partial W} = (X^1)^{Tp} \cdot \sigma'(H^1)$	$\frac{\partial H^2}{\partial U} = \sigma'(H^2) \odot H^1$		(11)
$substituting\ (10)\ and\ (11)\ into\ (9)$ $\frac{\partial E}{\partial W} = \sum_{t=1}^{T} (X^1)^{Tp} \cdot (y^t - \hat{y}^t) \odot (\sigma'(\hat{y}^t) \odot V) \odot \overbrace{(\sigma'(H^t) \odot U) \odot \cdots (\sigma'(H^2) \odot U)}^{t-1\ times} \odot \sigma'(H^1)$ $\frac{\partial E}{\partial U} = \sum_{t=2}^{T} (y^t - \hat{y}^t) \odot (\sigma'(\hat{y}^T) \odot V) \odot \overbrace{(\sigma'(H^T) \odot U) \odot \cdots (\sigma'(H^3) \odot U)}^{t-2\ times} \odot \sigma'(H^2) \odot H^1$			(12)

3.1.3 Update Weights

All new recurrent weight, input weight, and output weight matrices are computed by:

$$W(new) = W(old) - \lambda \odot \frac{\partial E}{\partial W} W(old),\quad U(new) = U(old) - \lambda \odot \frac{\partial E}{\partial U} U(old),$$

$$V(new) = V(old) - \lambda \odot \frac{\partial E}{\partial V} V(old)\ ;\quad where\ \lambda = learning\ rate.$$

3.2 Application of RNN

Input vectors are of size 7 and contain each day's 7 features/factors such as opening prices, highest and lowest prices of the day, volume traded, moving average of past 5 days and momentum. Weight matrix is of size 7×15. Predicted output(future stock price prediction) is a scalar. Learning rate is set equal to 0.05.

Each day either 2 shares are sold if the future predicted price is lower, or, 2 shares are bought if the future predicted price is higher.

As new stock price data from year 2018 appears it can also be incorporated into the model in real-time to make more accurate predictions as RNN can process input vectors of variable sizes.

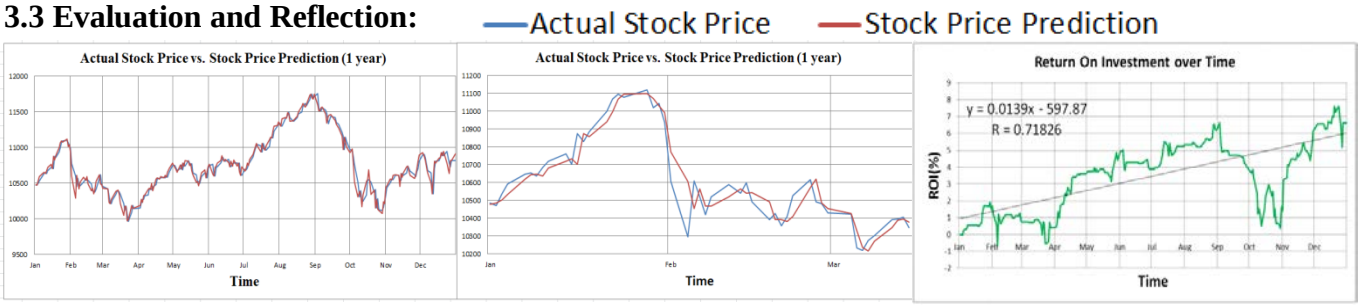
```
Day: 241    future prediction: 10726.675781     actual value:10817.900391
Day: 241    Not enough reserves to buy     share_no: 8    reserves:20072.210938

Day: 242    future prediction: 10819.425781     actual value:10820.950195
Day: 242    Not enough reserves to buy     share_no: 8    reserves:20072.210938

Day: 243    future prediction: 10819.385742     actual value:10913.200195
Day: 243    buy @ 10820.950195    share_no: 6    reserves:41714.109375

6 excess stocks
Total Returns: 107193.312500          ROI= 7.193311%
```

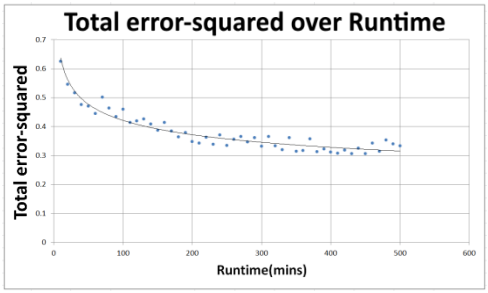
Console logs snippet

```
mulu = np.dot(U, new_input)
mulw = np.dot(W, prev_s)
add = mulw + mulu
s = sigmoid(add)
mulv = np.dot(V, s)
prev_s = s
```

Code snippet

3.3 Evaluation and Reflection:





RNNs are better for time-based relationships as past data effects future predictions. Return on investment(ROI) is higher at 7.19%. Standard deviation: 75.72 is significantly lower than DANN. Correlation coefficient of the trendline and actual returns is higher at 0.718 which may suggest that investments based on RNN's predictions are lower risk. Total error-squared of the optimized RNN is lower than DANN at ≈0.242. Regression time(time to optimize

weights) is lower at ≈400 minutes. Takes less time for regression as only 3 types of weight matrices have to optimized. Because of this, more no. of factors may be taken into account.

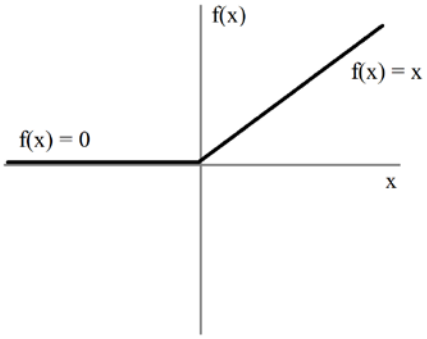
While these predictions appear to be very accurate, it is not necessarily so. As RNN incorporates and optimizes the model in real-time as new data of 2018 appears, its predictions are usually delayed by one day, reducing returns. Hypothetical eg: If Nifty 50 index falls by 100 units the next day, the RNN may predict incorrectly, but then optimize its strategy to reduce its stock price predictions for the next day. Even though the two curves appear very close, this reduces profits.

3.3.1 Vanishing gradient problem:

RNN cannot take into account long-term trends or long sequences. Long-term predictions are inaccurate and unreliable due to the vanishing gradient problem. As elements within U matrices are often < 1, past hidden matrices are multiplied by numbers < 1 repeatedly (in forward propagation). The hidden matrices of very old past time are not accounted for in the H^t matrix (current time calculations). $\lim_{x\to\infty} H^{t-x} \cdot U = 0$. The model overfit to current data and underfit to past data. Furthermore, This also implies that RNN cannot effectively model long sequences as when T approaches ∞ the effect of H^1, and consequently, input at time 1 have no consequence on H^T, and consequently, predictions at time T. This is a very big limitation as stock price data requires modelling of long sequences.

This is especially true when utilizing sigmoid activation function, as it often saturates weights to values close to 1 or 0. This often creates weight matrices where few neurons are doing the bulk of the calculations and rest just unnecessarily add on to the computations required, decreasing efficiency.

3.3.2 Solutions to Vanishing gradient problem:

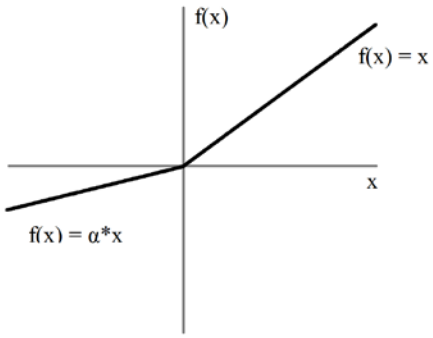


Another function used instead of sigmoid is defined as a hybrid function:

$ReLu(x) = \begin{cases} y = 0 \; for \; x < 0 \\ y = x \; for \; x > 0 \end{cases}$	$ReLu'(x) = \begin{cases} \frac{dy}{dx} = 0 \; for \; x < 0 \\ \frac{dy}{dx} = 1 \; for \; x > 0 \end{cases}$

This seems like an extremely simple function which makes the RNN more efficient and less computationally expensive. The derivatives are also simpler to compute. This may lead to drastically faster optimization of weights as the sigmoid function was calculated

Reduces the vanishing gradient problem as now the curve is linear and not concentrated near 1 and 0 (unlike sigmoid). This also creates sparser weight matrices compared to sigmoid with saturated weight matrices. One, of the drawbacks is that this function is not as complex as tanh or sigmoid and therefore may not accurately fit to the data when compared. Another limitation is that the function may ignore certain data points from the beginning of the dataset even though such data may be important as y = 0 when x < 0.



To solve this issue, gradient of y can be $0 < \alpha < 1$ which allows for some information to pass through the function when optimizing. This is known as leaky ReLu.

$Leaky \; ReLu(x) = \begin{cases} y = \alpha x \; for \; x < 0 \\ y = x \; for \; x > 0 \end{cases}$	$Leaky \; ReLu'(x) = \begin{cases} \frac{dy}{dx} = \alpha \; for \; x < 0 \\ \frac{dy}{dx} = 1 \; for \; x > 0 \end{cases}$

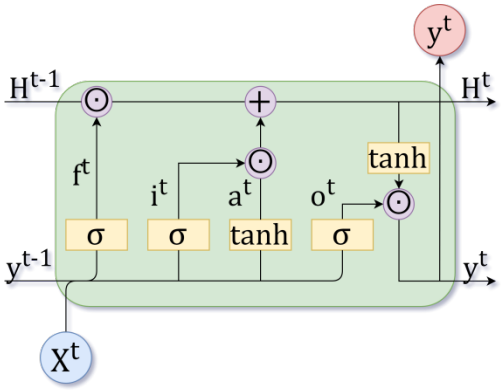
Section IV Long Short Term Memory (LSTM) Neural Networks (NN)

The vanishing gradient problem may be addressed even better by using LSTMNN as they can incorporate both long term and short term data to make effective predictions.

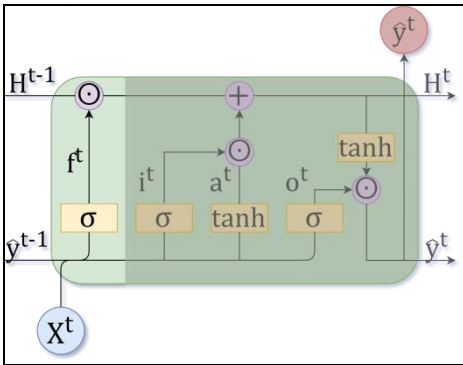
4.1: Understanding Long-short term memory neural networks

This model uses a similar concept to a neural network machine learning algorithm to train the model on how to store and replace data within a memory. LSTMNN then processes input data to give outputs based on this memory. This accounts for both, short-term and long-term data.

4.1.1 Forward Propogation:

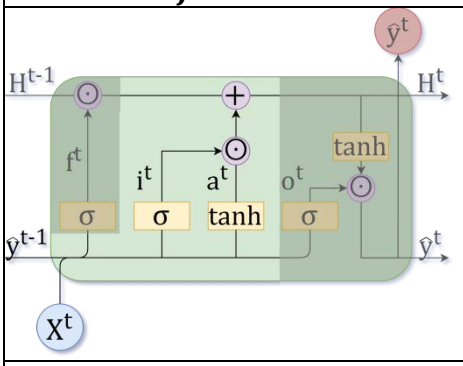


There are 4 layers at every time(t) f^t, i^t, a^t, o^t
U_f, U_i, U_a, U_o are all recurrent weight vectors of each layer.
W_f, W_i, W_a, W_o are all recurrent weight vectors of each layer.
b_f, b_i, b_a, b_o are all scalar biases of each layer.
H^{t-1} acts like the LSTMNN's memory.
H^0 *and* $\hat{y}^0$are null matrices. Time sequence starts from 1.
The breakdown of this diagram and its equations are given below:



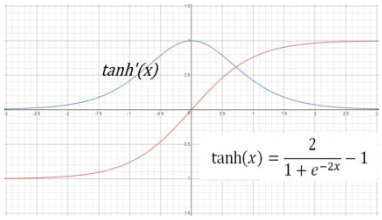
$$f^t = (W_f \cdot X^t + U_f \cdot H^{t-1} + b_f)$$
$$H^t = \boldsymbol{f^t \odot H^{t-1}} + a^t \odot i^t$$

"Forget layer"(f^t) decides what information is to be discard from the Hidden matrix. It gathers information from previous time predicted output($\hat{y}^t$), and the current time input(X^t) and outputs a scalar between 0 and 1 using the sigmoid activation function. The output of this layer decides how much of the information is to be kept (allowed to influence the rest of the subsequent layers). Previous time hidden matrix(H^{t-1}) is multiplied by f^t, forgetting the information as necessary.
In stock prediction, LSTMNN may want to discard unnecessary old information, while still keeping important old information intact.

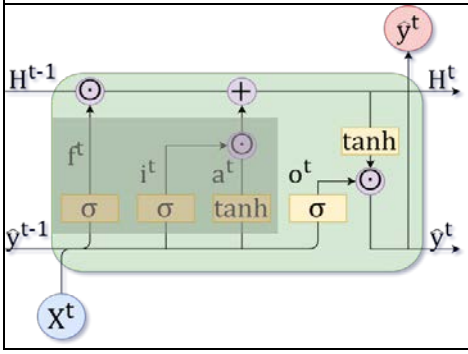


$$i^t = (W_i \cdot X^t + U_i \cdot H^{t-1} + b_i)$$
$$a^t = (W_a \cdot X^t + U_a \cdot H^{t-1} + b_a)$$
$$H^t = f^t \odot H^{t-1} + \boldsymbol{a^t \odot i^t}$$

"Input layer"($a^t \odot i^t$) decides to what extent the current time input and previous time output update the Hidden matrix with new information. $a^t \odot i^t$ is **added** to form the hidden matrix, and **not multiplied**. Due to this, no information is lost, only added.



The reason why tanh activation function is used, instead of sigmoid, is because tanh(x) is able to output a logistic value between -1 and 1, and therefore, can also account for, negative input values.
In stock price prediction, this layer processes how the new stock price information is added to the hidden matrix.



The filtered information from above set of operations is the used to calculate outputs. The hidden matrix is processed through tanh and then scaled by the output layer to calculate the predicted outputs.

$$o^t = (W_o \cdot X^t + U_o \cdot H^{t-1} + b_o)$$
$$H^t = f^t \odot H^{t-1} + \boldsymbol{a^t \odot i^t}$$
$$\hat{y}^t = \tanh(H^t) \odot o^t$$

4.1.2 Back Propogation:

Similar to RNN back propagation, but now, 8 weight vectors and 4 biases have to be optimized as follows:

Notation: $\theta = o, i, a, or\ f$. T is the largest time-step in the LSTM sequence. $\infty > T \geq t > 0\ and\ \ T, t \in N$

Total Error squared: $E = \sum_{t=1}^{T} \frac{1}{2}(y^t - \hat{y}^t)^2$ Error squared $\varepsilon^t = \frac{1}{2}(y^t - \hat{y}^t)^2$

$$\frac{\partial E}{\partial W_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t-1\ times} \frac{\partial \hat{y}^1}{\partial W_\theta} \quad \frac{\partial E}{\partial U_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^3}{\partial \hat{y}^2}}^{t-2\ times} \frac{\partial \hat{y}^2}{\partial U_\theta} \quad \frac{\partial E}{\partial b_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t-1\ times} \frac{\partial \hat{y}^1}{\partial b_\theta} \quad (13)$$

$$\frac{\partial \varepsilon^t}{\partial \hat{y}^t} = \frac{\partial}{\partial \hat{y}}\left(\frac{1}{2}(y^t - \hat{y}^t)^2\right) = (y^t - \hat{y}^t) \quad (14)$$

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$using\ quotient\ rule$$

$$tanh'(x) = \frac{-(e^x - e^{-x})(e^x - e^{-x}) + (e^x + e^{-x})(e^x + e^{-x})}{(e^x + e^{-x})^2} = \frac{-(e^x - e^{-x})^2 + (e^x + e^{-x})^2}{(e^x + e^{-x})^2} = \frac{\cancel{(e^x + e^{-x})^2}\left(1 - \frac{e^x - e^{-x}}{e^x + e^{-x}}\right)^2}{\cancel{(e^x + e^{-x})^2}} = 1 - tanh^2(x) \quad (15)$$

$substituting\ (15), using\ product\ rule \quad \frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial \hat{y}^{t-1}}$

$$= \left(\frac{\partial \hat{y}^t}{\partial o^t} \frac{\partial o^t}{\partial \hat{y}^{t-1}}\right) + \left(\frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial i^t} \frac{\partial i^t}{\partial \hat{y}^{t-1}}\right) + \left(\frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial a^t} \frac{\partial a^t}{\partial \hat{y}^{t-1}}\right) + \left(\frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial f^t} \frac{\partial f^t}{\partial \hat{y}^{t-1}}\right)$$

$$= tanh(H^t)\ \odot\ ((o^t) \odot (1 - (o^t))) \odot U_O$$
$$+ o^t\ \odot\ (1 - tanh^2(H^t)) \odot a^t \odot ((i^t) \odot (1 - (i^t))) \odot U_i$$
$$+ o^t \odot (1 - tanh^2(H^t)) \odot i^t \odot (1 - ((a^t)^2)) \odot U_a$$
$$+ o^t \odot \left(1 - tanh^2(H^t)\right) \odot H^{t-1} \odot ((f^t) \odot (1 - (f^t))) \odot U_f$$
$$= \delta \hat{y}^t \quad (16)$$

$$\frac{\partial \hat{y}^t}{\partial W_O} = \frac{\partial \hat{y}^t}{\partial o^t} \frac{\partial o^t}{\partial W_O} = tanh(H^t) \odot ((o^t) \odot (1 - (o^t))) \odot X^t = \delta W_O$$
$$\frac{\partial \hat{y}^t}{\partial W_i} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial i^t} \frac{\partial i^t}{\partial W_i} = o^t\ \odot\ (1 - tanh^2(H^t)) \odot a^t \odot ((i^t) \odot (1 - (i^t))) \odot X^t = \delta W_i$$
$$\frac{\partial \hat{y}^t}{\partial W_a} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial a^t} \frac{\partial a^t}{\partial W_a} = o^t \odot (1 - tanh^2(H^t)) \odot i^t \odot (1 - ((a^t)^2)) \odot X^t = \delta W_a$$
$$\frac{\partial \hat{y}^t}{\partial W_f} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial f^t} \frac{\partial f^t}{\partial W_f} = o^t \odot \left(1 - tanh^2(H^t)\right) \odot H^{t-1} \odot ((f^t) \odot (1 - (f^t))) \odot X^t = \delta W_f \quad (17)$$

$$\frac{\partial \hat{y}^t}{\partial U_O} = \frac{\partial \hat{y}^t}{\partial o^t} \frac{\partial o^t}{\partial U_O} = tanh(H^t) \odot ((o^t) \odot (1 - (o^t))) \odot \hat{y}^{t-1} = \delta U_O$$
$$\frac{\partial \hat{y}^t}{\partial U_i} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial i^t} \frac{\partial i^t}{\partial U_i} = o^t\ \odot\ (1 - tanh^2(H^t)) \odot a^t \odot ((i^t) \odot (1 - (i^t))) \odot \hat{y}^{t-1} = \delta U_i$$
$$\frac{\partial \hat{y}^t}{\partial U_a} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial a^t} \frac{\partial a^t}{\partial U_a} = o^t \odot (1 - tanh^2(H^t)) \odot i^t \odot (1 - ((a^t)^2)) \otimes \hat{y}^{t-1} = \delta U_a$$
$$\frac{\partial \hat{y}^t}{\partial U_f} = \frac{\partial \hat{y}^t}{\partial H^t} \frac{\partial H^t}{\partial f^t} \frac{\partial f^t}{\partial U_f} = o^t \odot \left(1 - tanh^2(H^t)\right) \odot H^{t-1} \odot ((f^t) \odot (1 - (f^t))) \otimes \hat{y}^{t-1} = \delta U_f \quad (18)$$

$substituting\ (14), (16), and\ (17)\ into\ (13) \quad \frac{\partial E}{\partial W_\theta} = \sum_{t=1}^{T} (y - \hat{y})\ \overbrace{\odot \delta \hat{y}^T \cdots \odot \delta \hat{y}^2}^{t-1\ times}\ \odot \delta W_\theta$

substituting (14), (16), *and* (18) *into* (13)	$\frac{\partial E}{\partial U_\theta} = \sum_{t=2}^{T} (y - \hat{y}) \overbrace{\odot \delta\hat{y}^T \cdots \odot \delta\hat{y}^3}^{t-2\ times} \odot \delta U_\theta$	(19)
substituting (14), *and* (16) *into* (13)	$\frac{\partial E}{\partial b_\theta} = \sum_{t=1}^{T} (y - \hat{y}) \overbrace{\odot \delta\hat{y}^T \cdots \odot \delta\hat{y}^2}^{t-1\ times}$	

4.1.3: Update Weights

$$W_\theta(new) = W_\theta(old) - \lambda \odot \frac{\partial E}{\partial W_\theta} W_\theta(old), \quad U_\theta(new) = U_\theta(old) - \lambda \odot \frac{\partial E}{\partial U_\theta} U_\theta(old),$$

$$b_\theta(new) = b_\theta(old) - \lambda \odot \frac{\partial E}{\partial b_\theta} b_\theta(old) \ ; \quad where\ \lambda = learning\ rate.$$

Computed for all $\theta = o, i, a, and\ f$

4.2 Application of LSTM neural networks

Now, sentiment analysis was also incorporated as a factor. Sentiment analysis[12] analyses a text using a LSTM neural network, to predict whether the text has good sentiments (represented by predicted output of 1) or bad sentiments (represented by predicted output of 0). Sentiment analyses will be from Sentiments of webpage titles of top 10 search results of Google news + Yahoo Finance News for total summed value. Input vectors are of size 7 and contain each day's 7 features/factors such as opening prices, highest and lowest prices of the day, volume traded, moving average and momentum of past 5 days, and sentiment analysis.

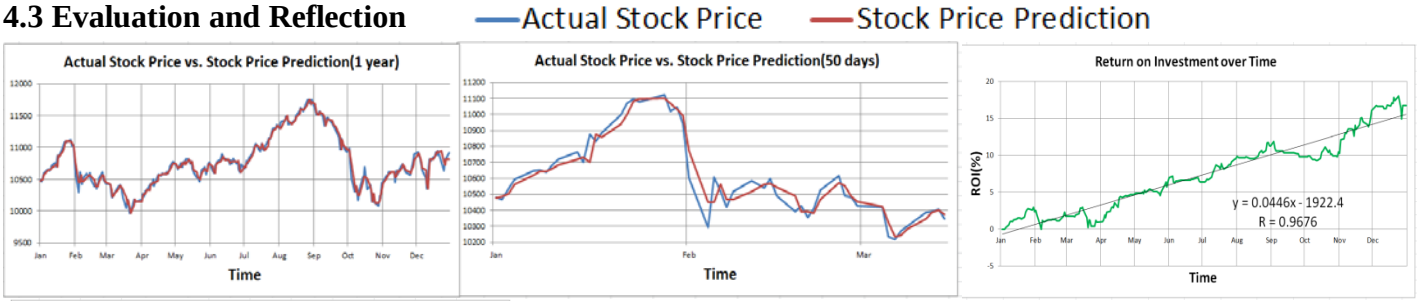
```
Day: 241    future prediction: 10726.675781      actual value:10817.900391
Day: 241    Not enough reserves to buy      share_no: 10     reserves:8283.011719

Day: 242    future prediction: 10819.425781      actual value:10820.950195
Day: 242    Not enough reserves to buy      share_no: 10     reserves:8283.011719

Day: 243    future prediction: 10819.385742      actual value:10913.200195
Day: 243    buy @ 10820.950195    share_no: 0     reserves:29924.912109

8 excess stocks
Total Returns: 117230.515625          ROI= 17.230515%
```

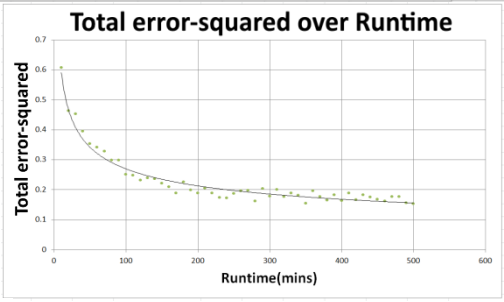
Console logs snippet

```
def sigmoid(X):
    return 1/(1+np.exp(-X))

def tanh_activation(X):
    return np.tanh(X)

def tanh_derivative(X):
    return 1-(X**2)

def sigmoid_derivative(X):
return (X(1-X))
```

Code snippet

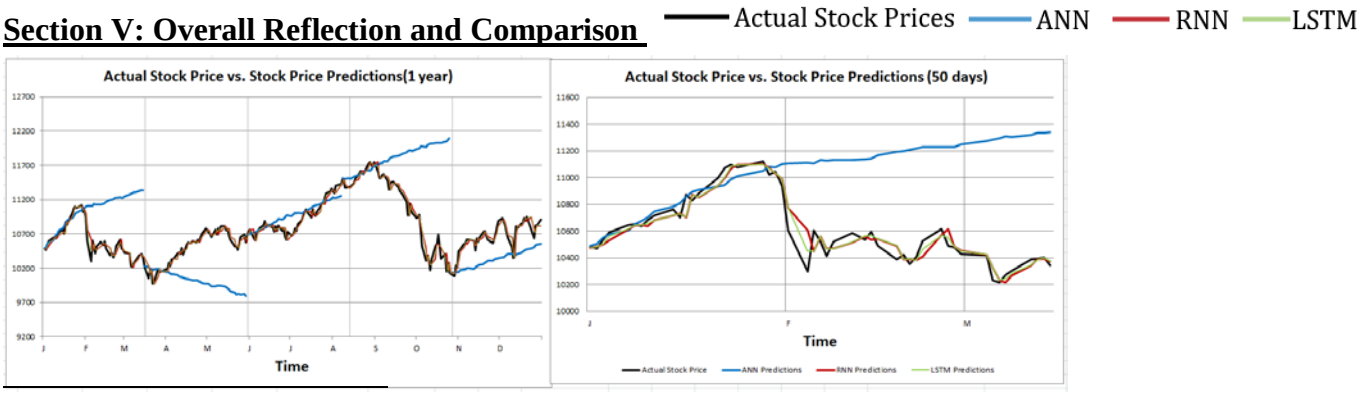
4.3 Evaluation and Reflection



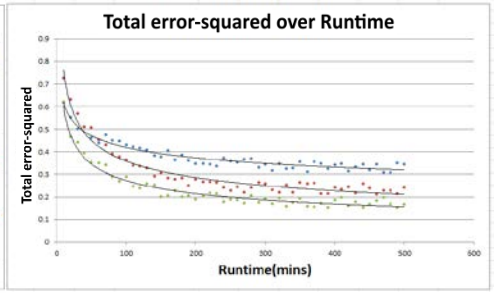


Return on investment is higher than RNN at 17.23%. Standard deviation: 65.97 is lower than RNN. LSTMs can take into account long-term trends as it does not get affected by vanishing gradient problem to the extent of RNN. Correlation coefficient of the trendline and actual returns is high at 0.967 which may suggest that investments based on LSTMNN's predictions are low risk. Total error-squared of the optimized LSTMNN is lower than RNN at ≈0.167. Regression time(time to optimize weights) is higher than RNN at 450 minutes. Requires more time for regression than RNN as there are 4 types of weights to be optimised. Because of this, less no. of factors may be taken into account as otherwise, the runtimes increase exponentially.

Section V: Overall Reflection and Comparison



[12] The process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic, product, etc. is positive, negative, or neutral. (Oxford Dictionary)



		DANN	RNN	LSTM
Return on Investment (measure of efficacy)	higher is better	0.16%	7.19%	17.23%
Standard deviation (measure of efficacy)	lower is better	565.19	75.72	65.97
Correlation coefficient of trendline with returns (measure of risk)	higher is better	0.014	0.71	0.967
error-squared when optimized(measure of efficacy)	lower is better	≈0.345	≈0.242	≈0.167
Time for optimization (measure of efficiency)	lower is better	≈450 minutes	≈400 minutes	≈450 minutes

5.1 Strengths

Model was optimized on 4 years of stock price data, and, the several types of weight matrices were of large enough size making the model more generalizable and effective in accurately and reliably predicting future stock prices. Many factors were taken into account for stock price prediction.

5.2 Limitations

Constant rate trading strategy of buying and selling 2 units/day(for LSTMNN and RNN) and 2 units/50 days(for DANN) is ineffective in producing returns on investment.

The ROI gained in year 2018 by usage of these models may not be an accurate representation of the norm.

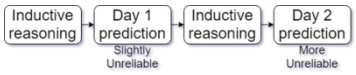
Not all factors that affect stock price were accounted for.

Only per data was available. However, intraday data may be used for more effective predictions and allow for shorter time period predictions.

The data of 4 years, while a large sample size, is still a sample, and therefore not representative of the whole.

The models only form sub optimal generalisations for the data. i.e. Predictions will not be 100% accurate.

Long-term predictions become increasingly inaccurate and unreliable as the model(which is already sub-optimal) is recursively applied, decreasing accuracy as time predicted increases.



5.3 Further areas of research

Model may be optimized on even more data and larger weight matrices/vectors may be utilised in order to make the model more accurate and extrapolatable in its predictions. More factors can be accounted for in the neural networks than the 7 used. The evaluation of the model may be conducted over a longer period of time. Optimized trading strategy of buying and selling more or less based on risks and rewards of the stock investment may be considered. In this case, Bayesian neural networks may be utilized that give probability distributions as predicted outputs while accounting for uncertainties.

Conclusion

To conclude, with the comparative analysis, LSTMs seem to be the best option for stock price prediction due to their ability to incorporate both short term and long term data. When compared to safe investments like fixed deposit, government bonds, and mutual funds, which usually have 6% to 12% mean ROI per year[13]. Investments based on LSTM's predictions are more effective in generating returns on investment at 17.23%. They may also be less risky as the fluctuations from this rate of ROI are low and investments are made in shorter periods of time, wherein stock prices usually fluctuate lesser than long-term investments. This IA has given me a greater appreciation for the applications of mathematics, especially discreet mathematics. I have found out that mathematics, through neural networks, can be applied to any field of studies, including commerce.

[13] *Indian Mutual Funds Handbook: A Guide for Industry Professionals and Intelligent Investors*, by Sundar Sankaran, Vision Books Pvt. Ltd., 2003, Accessed on 17 September, 2018.

Bibliography

There are many notations used within the IA that are used in computer science, especially, artificial neural networks' nomenclature which may seem to be plagiarised but are, in fact, the works of the author.

Tools used: CodeBlocks, Microsoft Excel, python, OpenMat, OpenMath, Symbolab, Desmos, WolframAlpha, Mathway, Photomath, Mathpix, GATE plugins, Paint.net, Snipping tool, LoggerPro, Microsoft word, etc.

"A Primer on how to optimize the Learning Rate of Deep Neural Networks", by Timo Böhm, *Towards Data Science*, https://towardsdatascience.com/learning-rate-a6e7b84f1658 Accessed on: 19 August, 2018.

"Activation Functions in Neural Networks", by Sagar Sharma, *Towards Data Science*, Sep 6, 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 Accessed on: 4 September, 2018.

"Backpropogating an LSTM: A Numerical Example", by Aidan Gomez, *Medium*, Apr 19, 2016. https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9 Accessed on:19 September, 2018.

"Definition of Outer Product", *Chegg Study*, 2018. https://www.chegg.com/homework-help/definitions/outer-product-33 Accessed on: 7July, 2018.

"Estimating an Optimal Learning Rate for a Deep Neural Network", by Pavel Surmenok, *Towards Data Science*, Nov 13, 2017 https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0 Accessed on: 17 August, 2018.

"How to Multiply Matrices," *Advanced Math is Fun*, 2017, https://www.mathsisfun.com/algebra/matrix-multiplying.html Accessed on: 4 August, 2018.

Indian Mutual Funds Handbook: A Guide for Industry Professionals and Intelligent Investors, by Sundar Sankaran, Vision Books Pvt. Ltd., 2003, Accessed on 17 September, 2018.

"Learning Rate Tuning and Optimizing", by Chaitanya Kulkarni, *Medium*, Feb 19, 2018, https://medium.com/@ck2886/learning-rate-tuning-and-optimizing-d03e042d0500 Accessed on: 23 August, 2018.

"Machine Learning week 1: Cost Function, Gradient Descent and Univariate Linear Regression", by Lachlan Miller, *Medium*, Jan 10, 2018 https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd Accessed on: 11 July, 2018.

The Matrix Cookbook by Kaare Brandt Petersen, Michael Syskind Pedersen, January 5, 2005https://www.ics.uci.edu/~welling/teaching/KernelsICS273B/MatrixCookBook.pdf Accessed on 9 October, 2018

"Neural Networks Demystified [Part 2: Forward Propagation]", by Welch Labs, *Youtube*, Nov 7, 2014. https://www.youtube.com/watch?v=UJwK6jAStmg Accessed on: 30 July, 2018.

"Neural Networks Demystified [Part 3: Gradient Descent]" by Welch Labs, *Youtube*, Nov 21, 2014. https://www.youtube.com/watch?v=5u0jaA3qAGk Accessed on: 30 July, 2018.

"Only Numpy: Deriving Forward feed and Back Propagation in Long Short Term Memory (LSTM) part 1", by Jae Duk Seo, *Towards Data Science*, Jan 12, 2018. https://towardsdatascience.com/only-numpy-deriving-forward-feed-and-back-propagation-in-long-short-term-memory-lstm-part-1-4ee82c14a652 Accessed on: 15 July, 2018.

"The Artificial Neural Networks handbook: Part 1", produced by Jayesh Bapu Ahire, *Data Science Central*, August 24, 2018 . https://www.datasciencecentral.com/profiles/blogs/the-artificial-neural-networks-handbook-part-1 Accessed on: 5 September, 2018.

"Understanding LSTM Networks", by Felix Gers, et.al., *Colah's Blog*, August 27, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/ Accessed on: 5 September, 2018. 21 July, 2018.

“What is the sigmoid function, and what is its use in machine learning's neural networks? How about the sigmoid derivative function?” by Vinay Kumar R *Quora,* Aug 24, 2017. https://www.quora.com/What-is-the-sigmoid-function-and-what-is-its-use-in-machine-learnings-neural-networks-How-about-the-sigmoid-derivative-function Accessed on: 7 September, 2018.