Applications of Machine Learning in Sequence Prediction Tasks

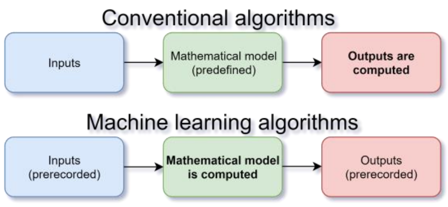
Siddharth Agrawal, Shaan Sheth, Shrey Sheth, Aarya Shah

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, Artificial Neural Networks, Recurrent Neural Networks, and Long Short-Term Memory are discussed in this paper, with reference to their applications in stock price prediction and parts of speech tagging. Furthermore, a new approach for an automatic trading algorithm is discussed.

Index Terms—Machine Learning, Hidden Markov Models, Artificial Neural Networks, Recurrent Neural Networks, Long Short-Term Memory, sequence prediction

Introduction:



Machine learning is a branch of 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. They often need to be trained on a large amount of data.

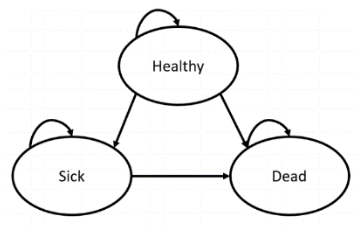


Figure 1: A simple Depiction of a Markov Model Bounthavong, M. (2018, March 15).

Markov Chains are a type of stochastic process that use simple mathematical and statistical models and make predictions from the current data. They are a way to statistically model a random process such as to predict the weather tomorrow given that the weather is sunny or cloudy today. They consist of a set of probabilistic distributions that satisfies the Markov Property, which means that they are *memoryless* or they cannot take into account the entire set of previous data and will look at only the current scenario of events to give us the prediction of the future. [2]

Artificial neural networks are a type of machine learning algorithm. These neural networks use repeated dot product multiplications and matrix additions, along with nonlinear activation functions applied on these matrices, in order to get better predictions. In this investigation, we will be going over a few machine learning models: Hidden Markov Model, Deep Artificial neural networks, recurrent neural networks, and long-short term memory neural networks. Each of these models acts as a basis for understand the next, more complex

[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.
[2] Bounthavong, M. (2018, March 15). Mark Bounthavong. Mark Bounthavong. https://mbounthavong.com/blog/2018/3/15/generating-survival-curves-from-study-data-an-application-for-markov-models-part-2-of-2

model. Mathematics of the all of these models will be explored in depth, and then the models will be applied for the specific case study of stock price prediction of Nifty-50 and Adani Green, and parts of speech tagging.

Background:

Markov Chains were first studied and developed by Andrey Markov by sifting through a bunch of vowels and consonants. He published his theory and findings in 1913 and they later came to be known as Markov Chains and paved the way for a whole new area of probability and statistics to be worked upon and discovered. It went far beyond what was known up till then in the field. Later Andrey Kolmogrov developed equations to predict the n^{th} timestep of a Markov Chain.

The history of Neural Networks is long and complicated, dating back 100s of years. Many researchers have constantly innovated on neural networks and continue to do so. Ruineihart, D., & Williams, R. first outlined the idea of backprogating error through time, which later built the foundation for recurrent neural network and similar architectures[3]. Hochreiter, S., & Schmidhuber, J. first developed LSTM in 1997[4], which was a form of gated RNN that could retain memory for a longer time. Since then, there have been several different conceptualisations of gated neural networks such as LSTM and GRU, each slightly different.

Motivation:

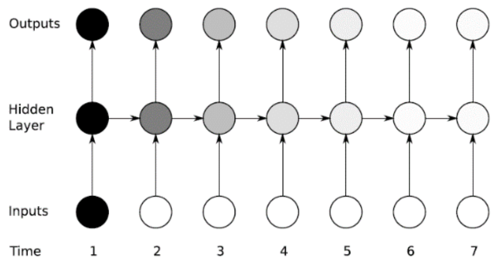
While plenty of literature and pretrained models as well as architectures are available in abundance. The in-depth mathematical and computational understanding of models used in sequential data prediction is lacking. This paper will be demystifying these models by delving deep into the mathematics and computations behind the ML models used for such applications. The paper will cover the basic ML implementations such as Hidden Markov Models, Artificial Neural Networks, Recurrent Neural Networks, and Long Short-term memory Neural Networks.

Furthermore, much of the literature on stock price prediction only focuses on predictions derived via mean squared error. This however, is insufficient if one actually wants to profit from stock price prediction models. Therefore, we innovate on the conventional mean squared error function, by proposing a new error function that optimises the amount of money earned over a trading period.

Literature Survey:

Conventional strategies of stock prediction such as news, buy and hold, martingales, momentum, mean reversion, technical analysis, 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.



[3] Ruineihart, D., & Williams, R. (n.d.). LEARNING INTERNAL REPRESENTATIONS BERROR PROPAGATION two. https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf
[4] Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(1), 1–42. https://doi.org/10.1162/neco.1997.9.1.1

To address vanishing gradients problem, many models now use LSTM networks. Such networks have longer memory than RNN networks[5].

“Stochastic Processes” book by Sheldon M. Ross was used to understand and draft equations relating to the Hidden Markov Model.[6]

YouTube video “Neural Networks Demystified” [7], and an article by Prof. Adian Gomez going over a numerical example of LSTM[8], going over the calculus, statistics, and linear algebra behind these models and how computations are carried out. This is especially useful for the backpropagation computations. We used this information to draft a model and generalised equations describing the ANN, RNN and LSTM models.

[5] Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(1), 1–42. https://doi.org/10.1162/neco.1997.9.1.1

[6] *Ross, Sheldon M. Stochastic Processes*. (2011). John Wiley & Sons, 1995. https://books.google.co.in/books/about/Stochastic_Processes.html?id=qiLdCQAAQBAJ&redir_esc=y

[7] “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, 2021.

[8] Aidan Gomez. (2016, April 18). *Backpropogating an LSTM: A Numerical Example - Aidan Gomez - Medium*. Medium; Medium. https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9

Contributions:

- **Shrey Sheth:**
  - Hidden Markov Model (HMM):
    - Introduction to Markov Chains
    - Background history of Markov Chains
    - Applications of Matkov Chains
    - Markov Chains and Machine Learning
    - Mathematical definition and transition matrices of HMM
    - Stock market prediction of Nifty-50 code using HMM
- **Aarya Shah:**
  - Hidden Markov Model:
    - Part of Speech (POS) tagging using HMM
      - Code
      - Report
      - visualisations
      - Tables
- **Shaan Sheth:**
  - Hidden Markov Model:
    - Part of Speech (POS) tagging using HMM
      - Code
      - Report
      - Visualisations
      - Tables
  - Note: Shaan Sheth first had an accident during Diwali and later had Chikungunya.
- **Siddharth Agrawal:**
  - Matrix Operations
  - Hidden Markov Model:
    - Chapman Kolmogrov Equations
    - Algorithm for eigenvector/steady-state vector
    - Code for visualisations for stock market prediction
  - Artificial Neural Network
    - Forward propagation mathematical model
    - Back propagation mathematical model
    - Application to Adani Green stock market and visualisations
  - Recurrent Neural Network
    - Forward propagation mathematical model
    - Back propagation mathematical model
    - Application to Adani Green stock market and visualisations
    - Vanishing Gradient problem
  - Long Short-Term Memory Neural Network
    - Forward propagation mathematical model
    - Back propagation mathematical model
    - Application to Adani Green stock market and visualisations
  - Report writing: Introduction, conclusion, limitations, etc. for the neural networks

Matrix Operations

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:

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

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:

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

Outer Product (symbol ⊗)

$$u \otimes v = uv^T$$
$$= \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$
$$= \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \vdots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix}$$ [9]

Note: Vectors may be thought of as an n × 1 matrix. Matrix and vector may be used interchangeably in some cases. As it is often easier to think and work with matrices, both in equations and computer memory.

[9] *Chegg.com*. (2021). Chegg.com. https://www.chegg.com/homework-help/definitions/outer-product-33

Section I Markov Chains:

Types of Markov Chains:

Markov Chains can be of various types, but on the basis of the type of time data, they are of two types. Discrete time Markov Chains and Continuous time Markov Chains.

In discrete time Markov Chains, the position of the object, also termed as the state of the Markov Chain in question, is recorded every unit of time, that is, t=1,2,3….. And so on. Whereas for continuous time model or version, the state is recorded at all times, that is, for any value of t>=0. An example for Discrete time Markov Chains would be Gambler's Ruin and one such example for Continuous time Markov Chains would be Brownian Motion and the Poisson Process.

Mathematical Definition and Transition Matrices:

Mathematically speaking, Markov Chains can be defined as follows:

The Markovian property is the property of stochastic process such that the next state depends only on the current state and none of the past states and that there is a fixed probability that if a process is in some state i then the probability of the process being in some other state j is denoted by P_{ij} where P is the transition probability matrix of the Markov chain.

$$P(X_{n+1} = j \mid X_n = i_n, X_{n-1} = i_{n-1}, \dots X_0 = i_0)$$

$$= P(X_{n+1} = j \mid X_n = i_n)$$

$$= P_{ij}$$

For positive n and all possible states ranging from $j,\ i_n,\ i_{n-1}, \dots\ i_0$ and all $n \geq 0$.

Which reiterates the definition discussed above that the Markov Chain is independent of the data from past instances and rather focuses on the current event.

Transition Matrix for a Markov Chain {X} at time t is the matrix containing all the information related to the probability of transition events or states. For a Matrix M with a state space S(i,j) where i and j are rows and columns respectively, the (i,j)th position is given by the formula,

M(i,j) = P(Xn= j | Xn-1= i)

The following is a hypothetical example of a transition probability matric of a process with two states:

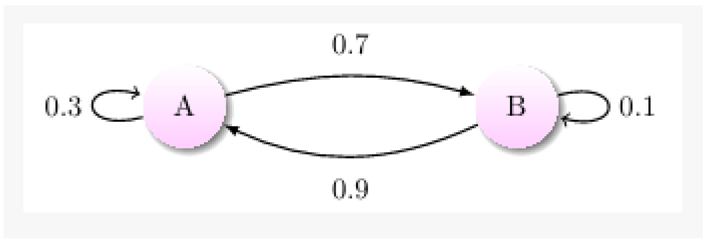


Figure 3: Time independent Markov Chain[10]

$$P = \begin{pmatrix} 0.3 & 0.7 \\ 0.9 & 0.1 \end{pmatrix}.$$

Figure 3: Probability Transition Matrix[10]

As is visible in Figure 3, the rows of the matrix are probability vectors with the sum of the row being 1 for each row as total probability always sums up to 1.

Chapman-Kolmogorov Equations:

Consider transition probability matrix at time n to be P_{ij}^{n} with s number of states.

$$P_{ij}^{n} = P(X_{n+m} = j \mid X_m = i)$$

If we want to find n-step transition probability matrix then we can do so by the following formula/proof:

$$P_{ij}^{n+m} = \sum_{k=0}^{s} P(X_{n+m} = j \mid X_0 = i)$$

$$P_{ij}^{n+m} = \sum_{k=0}^{s} P(X_{n+m} = j\ ,\ X_n = k \mid X_0 = i)$$

$$P_{ij}^{n+m} = \sum_{k=0}^{s} P(X_{n+m} = j \mid X_n = k,\ X_0 = i) P(X_n = k \mid X_0 = i)$$

Using the Markov property:

$$P_{ij}^{n+m} = \sum_{k=0}^{s} P(X_{n+m} = j \mid X_n = k) P(X_n = k \mid X_0 = i)$$

$$P_{ij}^{n+m} = \sum_{k=0}^{s} P_{ik}^{n} P_{kj}^{m}$$

10 *Markov Chains | Brilliant Math & Science Wiki*. (2021). Brilliant.org. https://brilliant.org/wiki/markov-chains/

Using this proof, let X denote the n-step transition probability matrix, and P denote the transition probability matrix, then n-step transition probability matrix, is P^n:

$$\mathrm{X} = P^n \cdot P^{n-1} = P^n \cdot P^{n-1} \cdot P^{n-2} = \cdots = P^n$$

This also means, that, given the initial marginal probability distribution as a vector (denoted by $\vec{\imath}$), we can get the n-step marginal probability distribution by calculating:

$$\vec{\imath} \cdot P^n$$

Markov chains' stationary distribution:

Given that a Markov chain is well connected (every state can go to other state with some probability > 0), then it converges to a stationary marginal probability distribution. We can find its stationary distribution by finding the eigenvector corresponding to eigenvalue 1 of the Markov chain transition matrix. This eigenvector is also called the steady state vector of a Markov chain.

Similarly, the n-step transition probability matrix also reaches convergence for large value of n.

$$\lim_{n\to\infty} P^n \approx P^n$$

Thus, the stationary distribution is also known as limiting distribution of a Markov chain.

Applications of Markov Chains:

Markov Chains, since the time they have been developed have found a number of applications in various places not just limited to Mathematical problems. A few of their applications are discussed below.

They are widely used in the hard sciences, that is, physics, chemistry and biology. They can be used to map enzyme activity and understand the state of chemical reactions, provided that the reaction follows a Markov Chain. Another application of Markov Chains can be found in bioinformatics where DNA evolution models make use of them to provide a description of the nucleotide present at a given site. They also find use in quantum mechanics to a huge degree.

They can also be used in areas apart from science such as economics and finance where they can be used to model the market trends and predict stock market crashes or predict recessions in the economy using macroeconomic principles. They can be used to calculate credit risks and model asset pricing as well. Markov Chains also find use in the Social Sciences, Weather forecasting, Video games, and many other fields.

Markov Chains in Machine Learning (Hidden Markov Models):

A statistical Markov model where the model is based on a Markov Chain with hidden states or unknown states is called a Hidden Markov Model (HMM). The main objective of the Hidden Markov Model is to learn more about the Markov Model by analyzing the hidden states within it.

An extremely simple example would be trying to find out the weather outside on any given day based on what the people outside or pedestrians are wearing. Here, in this example, the clothing of the pedestrians is the observed or known Markov Model or state and the weather is the unknown variable or Hidden Markov Model. In order to calculate the probability of the hidden variables, three types of data are necessary, transition data, emission data and initial state information. Transition data is as shown above in the matrix, the data that is the probability of transitioning to a new state based on the current state. Emission data is the probability of transitioning to an observed state based on the hidden state and finally, initial state data is the initial probability of transitioning to a hidden state.

Hidden Markov Models have quite a large number of applications in a varied range of areas such as computational finance, speech recognition, document separation in scanning solutions, handwriting recognition software, machine translation, transportation forecasting and many more.

Algorithm:

To find the eigenvector corresponding to eigenvalue of 1 of a Markov Chain transition probability matrix, we can use many methods. Two of the methods that apply well to this scenario will be discussed here:

Method 1: We can find vector x such that $(A - I)x = 0$ where A is a matrix and I is the identity matrix. This would require gaussian elimination and back substitution as the most efficient algorithm to solve the simultaneous system of linear equations with an algorithmic complexity of $O(n^3)$.

Method 2: Instead, we can also use power iterations method, which is a numerical method to find the largest eigenvalue and its eigenvector by repeated dot products of the matrix with any vector.

This method is preferable as it also gives the marginal probabilities for all timesteps starting from a particular state. And each step takes $O(n^2)$ and solutions converge linearly.

It can be proven that 1 is the largest eigenvalue of any stochastic process, including markov chain transition probability matrix:

Notation: $\vec{1}$ denotes a vector containing all 1s => $\begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$

Proof: If A is a transition probability matrix, then, since each row of A sums to 1. $A\vec{1} = \vec{1}$. Therefore, one of the eigenvalues is 1.

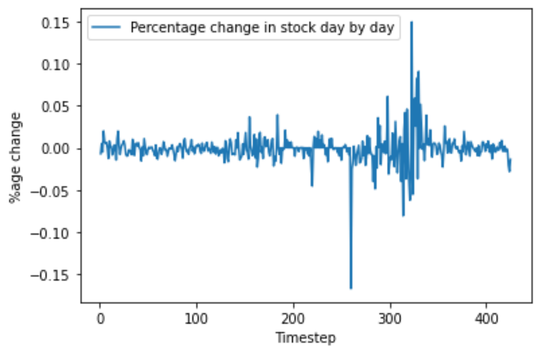
Assume there is some $\lambda > 1$ for which $A\lambda = \lambda\vec{x}$ is satisfied. Since the rows of A are nonnegative and add up to 1, each entry in vector $A\vec{x}$ must be a convex combination (a linear combination where coefficients are nonnegative and sum up to 1) of the elements of $\vec{x}$. This convex combination can be no greater than the largest value in $\vec{x}$. But at least one element in $\lambda\vec{x}$ will be greater than the largest value $\lambda\vec{x}$ if $\lambda > 1$. Therefore, this will dissatisfy the equality $A\lambda = \lambda\vec{x}$. Therefore, a transition probability matrix cannot have $\lambda > 1$.

Hence, power iteration method works for our scenario.

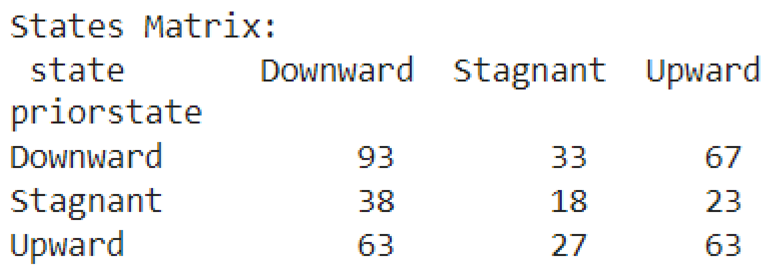
This method is preferable as it also gives the marginal probabilities for all timesteps starting from a particular state. This also reiterates the proof for Chapman Kolmogorov equations that were discussed, that a Markov chain converges to a stationary marginal probability distribution over time.

Application and Visualisations of Hidden Markov Model for Nifty-50 stock price prediction:

We first calculate the %age change in the stock prices over the last 500 days of Nifty-50 daily closing price data.



Then based on this, we make a Markov chain. If the percentage change is greater than 0.1% then we consider the stock price going up. If instead, the percentage change is less than -0.1%, then we consider it as the stock price going down. If instead, the percentage change stays between 0.1% and -0.1%, then we consider the market to have remained stagnant and not changed.



```
States Matrix:
 state        Downward  Stagnant  Upward
priorstate
Downward            93        33      67
Stagnant            38        18      23
Upward              63        27      63
```

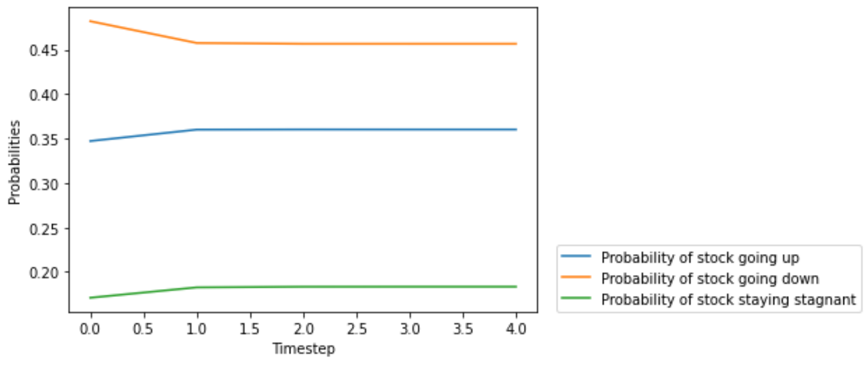
Figure 1: Transition Probability Matrix for 500 days of Nifty-50 data

Using this transition probability matrix, future timesteps are calculated by assuming the initial state to be downward trend, and then generating all future timesteps by recursively multiplying the initial marginal probability by the transition probability matrix:

Where $\vec{\imath}$ is the initial marginal probability distribution (in this case <1,0,0>), we can get the n-step marginal probability distribution by calculating:

$$\vec{\imath} \cdot P^n$$

Result:



The above plot illustrates the probability of the stock going up/down/remaining stagnant. We can see that the Markov chain results in a steady state vector:

```
[0.45646225 0.18338842 0.36014934]
```

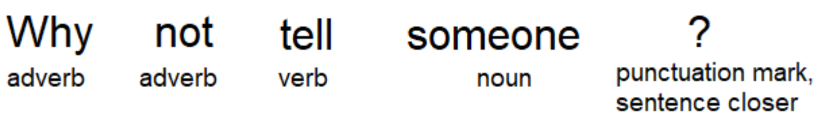
Where the first element is probability of the stock going down, 2nd element is the probability of the stock staying stagnant, and the third element is the probability of the stock going up.

We can see that the probability for down is more which would suggest that over the long term, more often than not, the next day's stock price will go down from the current day's stock price, though the amount of decrease in the stock price is not known using this model. So this may be applicable to something like binary options.

This model is fairly restrictive, as the future stock price depends on all previous timesteps, and not just the current timestep. Furthermore, it can only incorporate a single variable/factor, the closing stock price.

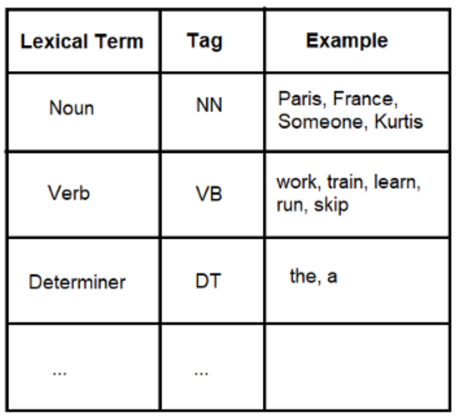
Parts of Speech Tagging:

Part of Speech tagging is a popular and widely NLP process where each word from a given corpus(text) is classified and categorised as different 'parts of speech' depending on its definition and context in the sentence. For eg.

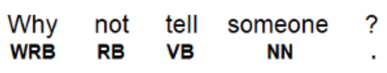


Why	not	tell	someone	?
adverb	adverb	verb	noun	punctuation mark, sentence closer

These categorisations for each term is known as a tagging and a specific tag is assigned to each part of speech. For eg.



Lexical Term	Tag	Example
Noun	NN	Paris, France, Someone, Kurtis
Verb	VB	work, train, learn, run, skip
Determiner	DT	the, a
...	...	



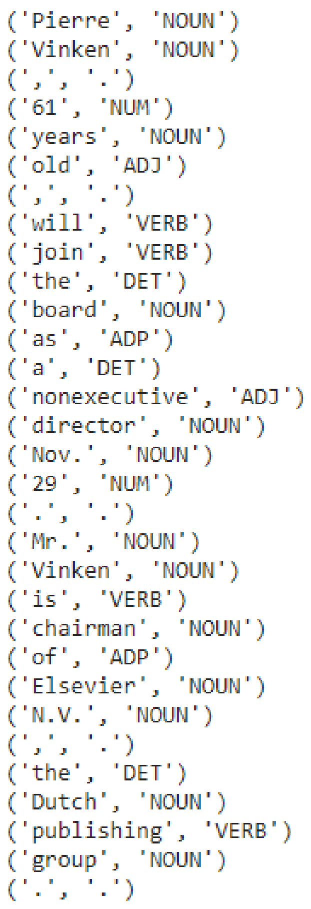
Why	not	tell	someone	?
WRB	**RB**	**VB**	**NN**	**.**

What the above procedure does is make it is easier for us to create a dataset from where in we can predict that in a given sentence what could the next part of speech be and hence, automatically complete a sentence enabling many further applications. This is done using HMM models and creating a probability transition matrix and an emission matrix. Once created, the entire model can be efficiently used for PoS tagging. We can describe markov models as "*a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the Markov property).*"[11]

There are 81 different kinds of PoS tags and hence creating a HMM model becomes very complex. To Optimise and implement a HMM, the Viterbi algorithm can be used. The code that we used for PoS tagging has 12 unique tags. As can. Be seen,

Code implementation

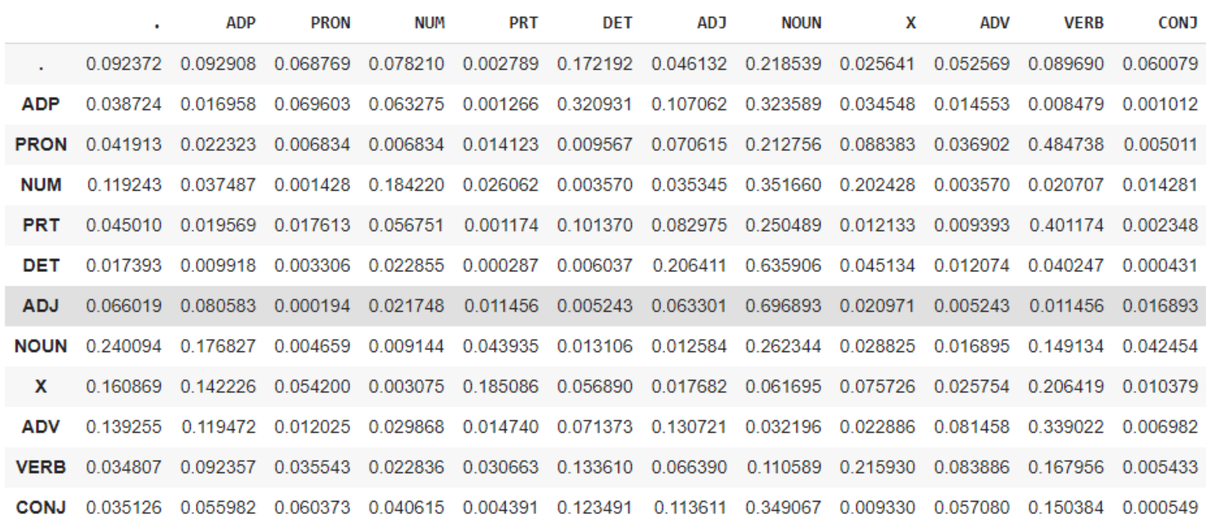
Given below is example of the tags used in the corpus.



```
('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
(',', '.')
('the', 'DET')
('Dutch', 'NOUN')
('publishing', 'VERB')
('group', 'NOUN')
('.', '.')
```

The below image represents the transition probability table.

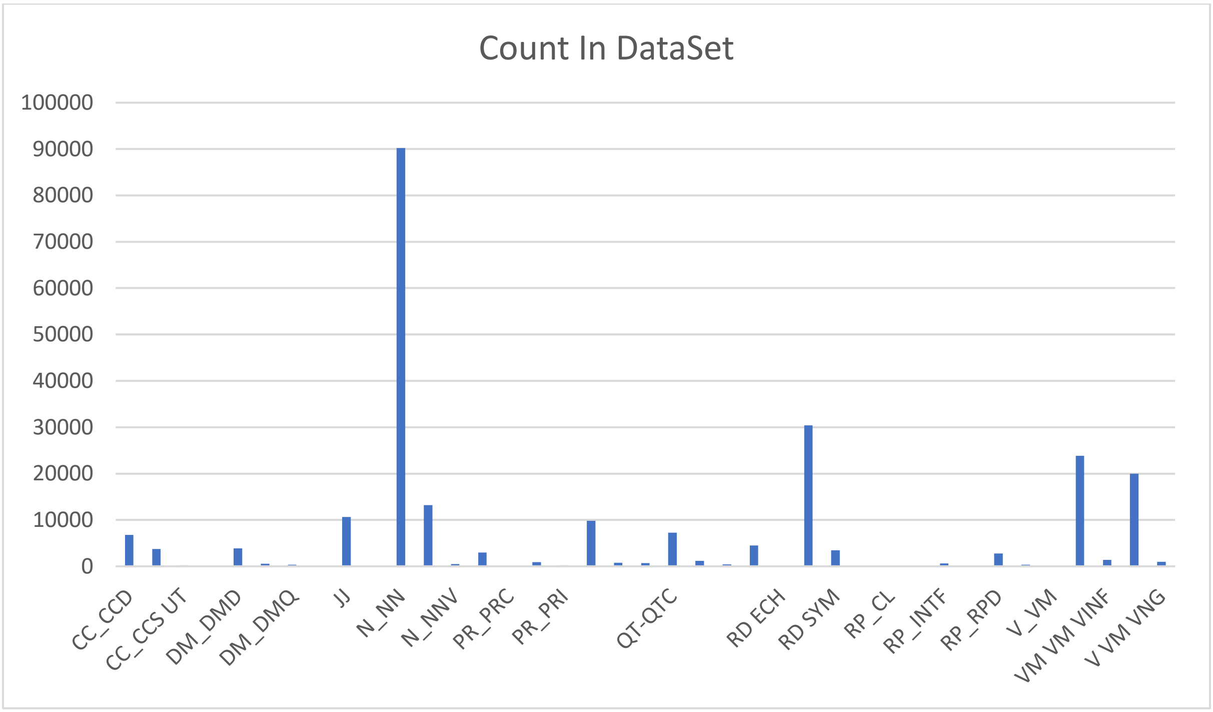
[11] Ryan, M. (2020, June 6). *A Very Simple Method of Weather Forecast Using Markov Model Lookup Table*. Medium; Towards Data Science. https://towardsdatascience.com/a-very-simple-method-of-weather-forecast-using-markov-model-lookup-table-f9238e110938



	.	ADP	PRON	NUM	PRT	DET	ADJ	NOUN	X	ADV	VERB	CONJ
.	0.092372	0.092908	0.068769	0.078210	0.002789	0.172192	0.046132	0.218539	0.025641	0.052569	0.089690	0.060079
ADP	0.038724	0.016958	0.069603	0.063275	0.001266	0.320931	0.107062	0.323589	0.034548	0.014553	0.008479	0.001012
PRON	0.041913	0.022323	0.006834	0.006834	0.014123	0.009567	0.070615	0.212756	0.088383	0.036902	0.484738	0.005011
NUM	0.119243	0.037487	0.001428	0.184220	0.026062	0.003570	0.035345	0.351660	0.202428	0.003570	0.020707	0.014281
PRT	0.045010	0.019569	0.017613	0.056751	0.001174	0.101370	0.082975	0.250489	0.012133	0.009393	0.401174	0.002348
DET	0.017393	0.009918	0.003306	0.022855	0.000287	0.006037	0.206411	0.635906	0.045134	0.012074	0.040247	0.000431
ADJ	0.066019	0.080583	0.000194	0.021748	0.011456	0.005243	0.063301	0.696893	0.020971	0.005243	0.011456	0.016893
NOUN	0.240094	0.176827	0.004659	0.009144	0.043935	0.013106	0.012584	0.262344	0.028825	0.016895	0.149134	0.042454
X	0.160869	0.142226	0.054200	0.003075	0.185086	0.056890	0.017682	0.061695	0.075726	0.025754	0.206419	0.010379
ADV	0.139255	0.119472	0.012025	0.029868	0.014740	0.071373	0.130721	0.032196	0.022886	0.081458	0.339022	0.006982
VERB	0.034807	0.092357	0.035543	0.022836	0.030663	0.133610	0.066390	0.110589	0.215930	0.083886	0.167956	0.005433
CONJ	0.035126	0.055982	0.060373	0.040615	0.004391	0.123491	0.113611	0.349067	0.009330	0.057080	0.150384	0.000549

Since in the table, there are no zero values, we can use It for further application as given tagging is correct .

Eg. Dataset and chart which can also be used in the code



Tag	Count In DataSet
CC_CCD	6772
CC_CCS	3700
CC_CCS UT	216
CL	16
DM_DMD	3849
DM_DMI	561
DM_DMQ	360
DM_DMR	1
JJ	10660

NN	7
N_NN	90220
N_NNP	13182
N_NNV	451
N_NST	2954
PR_PRC	15
PR_PRF	904
PR_PRI	241
PR_PRP	9830
PR_PRQ	755
PSP	720
QT-QTC	7244
QT_QTF	1208
QT_QTO	436
RB	4474
RD ECH	57
RD PUNC	30404
RD SYM	3481
RD UNK	1
RP_CL	5
RP INJ	166
RP_INTF	601
RP NEG	153
RP_RPD	2777
V_VAUX	355
V_VM	45
V_VM_VF	23849
VM VM VINF	1391
V_VM VNF	19986
V VM VNG	939

In our model which was trained on `treebank` and `universal_tagset` datasets, we had a very high accuracy on the validation/test dataset (dataset on which the model was not trained), therefore indicating that the model works very well and has high generalisability.

```
Time taken in seconds:  51.79842686653137
Viterbi Algorithm Accuracy:  91.41414141414141
```

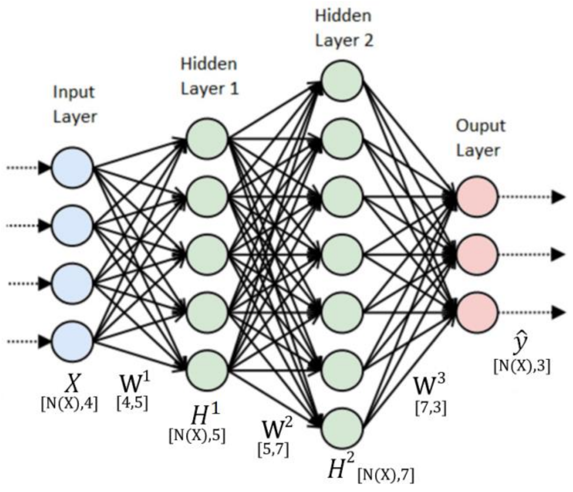
A more complex model with larger training dataset was used/can be used for predictive texting in mobile phones. Thus, we have explored, one of the ways in which Linear Algebra is applied in devices we use every day.

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.

Understanding Deep Artificial neural networks (DANN)

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:

$$\sigma(X \cdot W^1) = H^1$$

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

$$\sigma(H^N \cdot W^{N+1}) = \hat{y}$$

Here, $\sigma(x) = \frac{1}{1+e^{-x}}$ also known as the 'sigmoid function[12]' and this function is applied element-wise. N denotes the number of hidden states. 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.

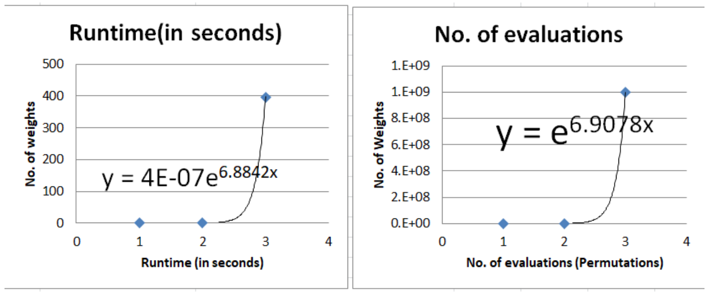
Back propagation

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).

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:

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

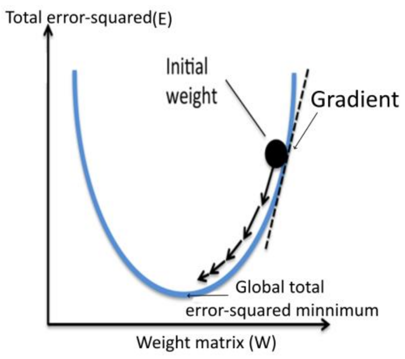
Let’s assume that only 1 weight value has to be optimized, and 1000 different weight values are tested. Using my computer, and a hypothetical dataset, this took 0.0004162 seconds. But as the number of weights increase, in order to maintain the same precision (1000), the runtimes increase exponentially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



This means that the complexity[13] 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

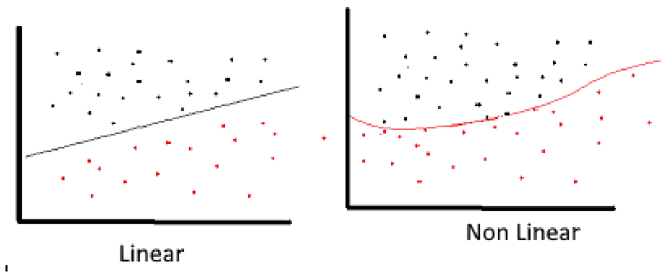
$= (4 \times 10^{-7})e^{6.8842(10)}$(Taken from figure to the left) (Time period longer than the universe’s existence).

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.

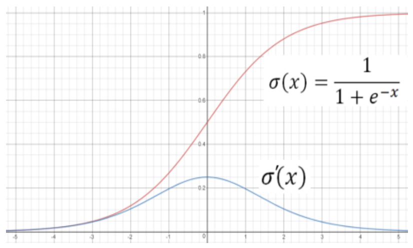


Note: partial derivative of total error-squared is taken with respect to each weight matrix as one weight matrix is taken into account at a time and other relationships are ignored.

The reason why activation function[14] like sigmoid is used is because of their non-linear property. This allows them to function better as universal function approximators. Furthermore, it can be seen that if we repeatedly apply matrix products to the original input, it would be no different than applying a single linear transformation to the input. The activation function allows for the several layers in the neural network to progressively filter and learn more about the inputs to outputs mapping as it is not equivalent to just a single linear transformation. To model the complex relationships in stock price prediction, a simple linear model would not work effectively and a non-linear model is required.



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



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 states, $\infty > N \geq n > 1$ *and* $N, n \in$ *Natural numbers*

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

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

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

using quotient rule,

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

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

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

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

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

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

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

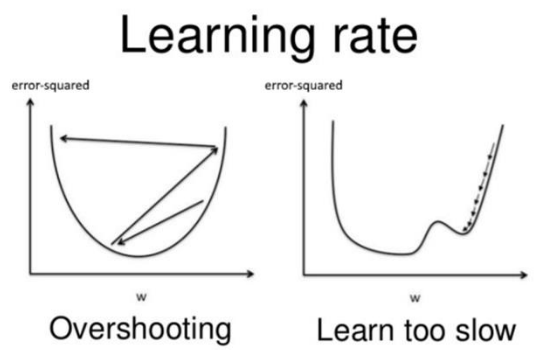
$$\frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial (H^N \cdot W^{N+1})} = \sum_{m=1}^{M} (y_m - \hat{y}_m) \odot \left(\sigma(\hat{y}_m) \odot (1 - \sigma(\hat{y}_m))\right) = \delta^N \qquad (4)$$

In back propagation of neural networks, some matrices may be transposed[15], 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 state. This is commonly used in matrix differentiation. The proofs and theorems for this can be found in the matrix cookbook as well.[16]	$\begin{bmatrix} \sum_{m=1}^{M} H_{m,1}^{N} \cdot \delta_m^{N+1} \\ \sum_{m=1}^{M} H_{m,2}^{N} \cdot \delta_m^{N+1} \\ \vdots \\ \sum_{m=1}^{M} H_{m,K}^{N} \cdot \delta_m^{N+1} \end{bmatrix} = (H^N)^{Tp} \cdot \delta^{N+1} = \frac{\partial E}{\partial W^N}$	(5)

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

Solving [2] by substituting (3). similar to (5)	$\frac{\partial(H^n \cdot W^{n+1})}{\partial H^n} \frac{\partial H^n}{\partial(H^{n-1} \cdot W^n)} = (W^{n+1})^{Tp} \odot \left(\sigma(H^n) \odot \left(1 - \sigma(H^n)\right)\right) = \delta^n$			(6)
Solving [3]	$\frac{\partial(H^{N-1} \cdot W^N)}{\partial W^N} = H^N$	$\frac{\partial(H^{n-1} \cdot W^n)}{\partial W^n} = H^{n-1}$	$\frac{\sigma(X \cdot W^1)}{\partial W^1} = X$	(7)
Substituting [1], [2] and [3] into (2), similar to (5), for final derivative as follows:				
$\frac{\partial E}{\partial W^N} = (H^N)^{Tp} \cdot \delta^{N+1}$	$\frac{\partial E}{\partial W^n} = (H^n)^{Tp} \cdot \overbrace{\delta^{N+1} \cdot \delta^{N} \cdots \delta^{n}}^{n+1\ times}$	$\frac{\partial E}{\partial W^1} = (X)^{Tp} \cdot \overbrace{\delta^{N} \cdot \delta^{N-1} \cdots \delta^{1}}^{N+1\ times}$		(8)

Update weights:



All new weights are computed by:

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

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

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.[17]

[15] A transposed matrix is a matrix whose original rows and columns are switched to obtain a new matrix.
[16] Brandt, K., Michael, P., & Pedersen, S. (n.d.). *The Matrix Cookbook*. https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf
[17] "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.

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.

Application of DANN:

First, all the data needs to be normalised by the min-max function:

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

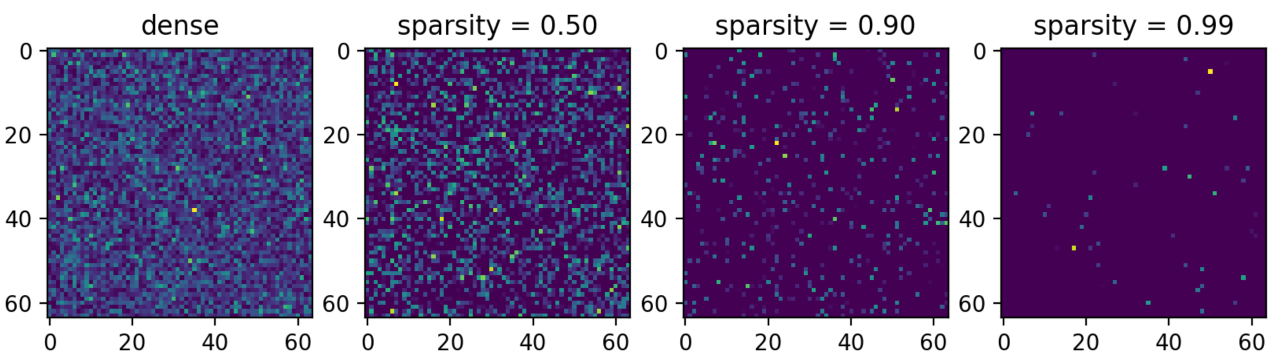
Where min is the lowest value in the dataset and max is the maximum value in the dataset. This scales all these values between 0 and 1.

Opening price	Normalized value
8210.10	0.250933
8196.05	0.247962
8202.65	0.249358

Figure 2: Hypothetical example of minmax scaling

This helps in two ways:

1. Numerical methods converge faster when the values are small. Furthermore, it reduces the risk of vanishing and exploding gradients. Vanishing gradients are when values in the weight matrices are so small that they do not contribute to the output. This leads to sparser matrices that use up computer resources while not performing significantly better. This also reduces the risk of values in the matrix exploding to a very large number that the computer cannot perform operations on.



[18]

Figure 3: Visual example of sparse matrices

2. The neural network relate between multiple factors well as they are now on the same scale. Otherwise, one factor can be several magnitudes larger than another, making it harder for the neural network to process the information as it has to first learn to scale up/down the factors. 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 Adani Green stock price data. This dataset works well as it is a respected and fairly new company on the stock exchange with good fundamentals and therefore has a small dataset on which the models can train faster. However, to optimise the model further, we would want to train the model on a large dataset of multiple stocks and then finetune it for a particular stock with a lower learning rate. This is known as transfer learning.

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):

$$minmax(x)\left(minmax^{-1}(x)\right) = f(x) \; = \; x.$$

As the main focus is on the models and the linear algebra, we used technical analysis library (`ta`) for python to generate 86 stock market indicators. Including momentum, volume, volatility, trend, and other indicators such as moving average, momentum, average true range, Bollinger bands, moving variance, etc. for each day. Of course, open, low, high, and closing stock prices of each day was also included in the dataset.

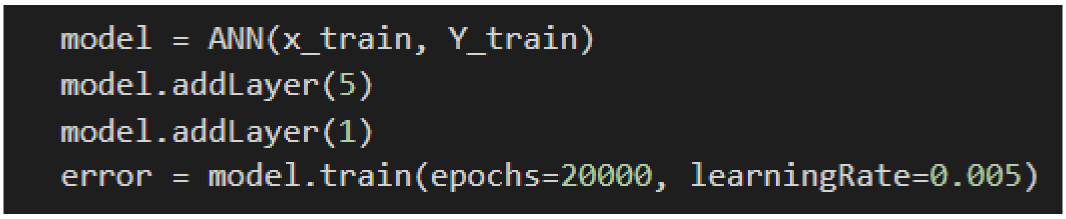
Full list of indicators can be found at the library's documentation website: https://technical-analysis-library-in-python.readthedocs.io/en/latest/ta.html. [19]

These were fed as the inputs to the ANN model and the output task for the model was to predict the next day's closing price. Ironically, we found that the ANN worked best when we added a single hidden layer of just size number of timesteps × 5, and two weight matrices (one mapping from input to hidden layer, another mapping from hidden layer to output). This is because it would train/converge much faster as it had fewer operations

[18] François Lagunas. (2020, February 4). *Is the future of Neural Networks Sparse? An Introduction (1/N)*. Medium; HuggingFace. https://medium.com/huggingface/is-the-future-of-neural-networks-sparse-an-introduction-1-n-d03923ecbd70

[19] bukosabino. (2021, November 20). *bukosabino/ta: Technical Analysis Library using Pandas and Numpy*. GitHub. https://github.com/bukosabino/ta

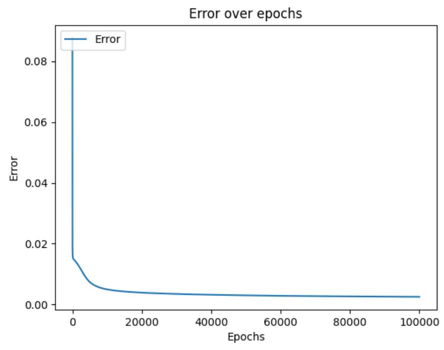
to perform per epoch. Learning rate was set to 0.005 and was trained for 20000 epochs. Other tests were also done during different parameters. Two such tests are included.

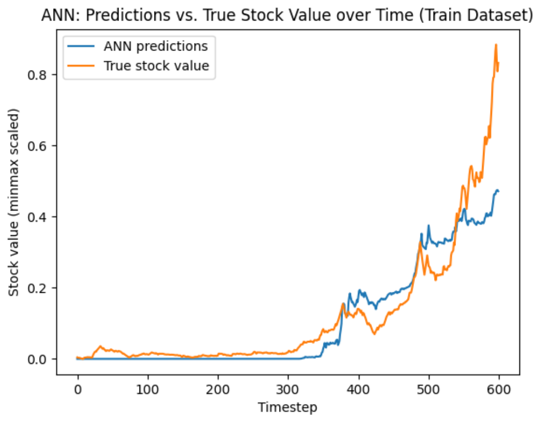


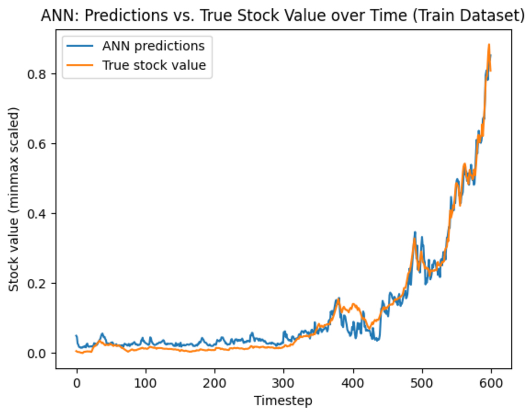
```
model = ANN(x_train, Y_train)
model.addLayer(5)
model.addLayer(1)
error = model.train(epochs=20000, learningRate=0.005)
```

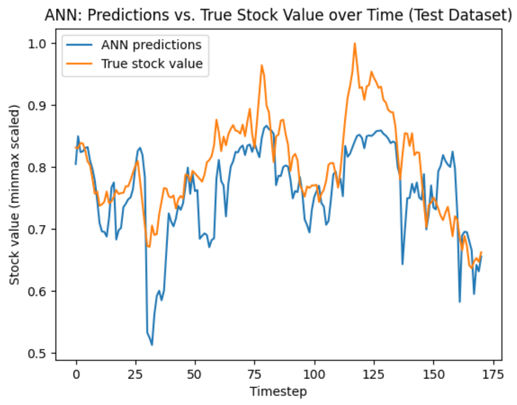
Figure 4: Snippet of instancing ANN model class. The ANN model class was coded by us from scratch with online code as reference

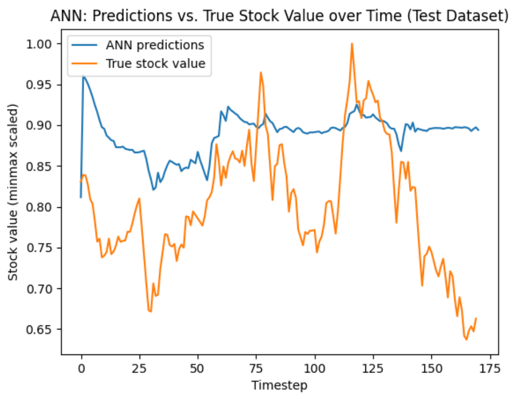
2.3 Outputs from code:











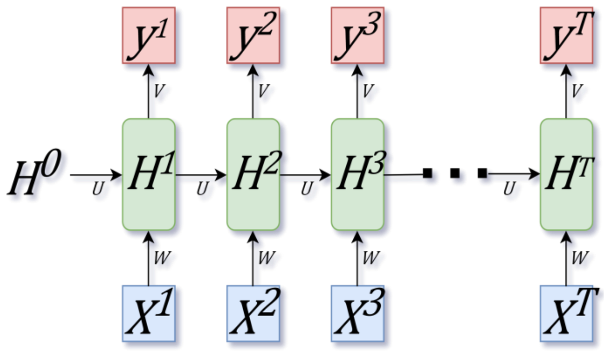
We can see that the predictions are very close to the actual value and the loss reduces over time which suggests that the model coded by us works as intended. The final mean squared error was 0.003971258200595761.

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.

Understanding Recurrent Neural Networks

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[20]. U, V, and W matrices are filled with samples from a continuous uniform distribution $U_{[0,1)}$.

Dimensions of W: input vector length × hidden vector length

Dimensions of U: hidden vector length × hidden vector length

Dimensions of Y: hidden vector length × output vector length

H, X and Y are assumed to be row vectors for our application.

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

$$a^t = W + H^{t-1} \cdot U$$

$$\tanh(X^t \cdot W + H^{t-1} \cdot U) = H^t$$
$$\Rightarrow \tanh(a^t) \qquad = H^t$$

$$H^t \cdot V = \hat{y}^t$$

The major difference between DANN and RNN is that $\mathit{Hidden\ matrix\ of}\ \boldsymbol{previous\ time}\ (H^{t-1}) \cdot \mathit{Recurrent\ weight\ matrix}(U)$ is **added** to the $\mathit{input\ of}\ \boldsymbol{current\ time}(X^t) \cdot \mathit{weight\ vector}(W)$. This allows the past data to have an effect on the current hidden state, and consequently output vector, and subsequent hidden states. 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 state and input vectors of **previous time(s)** affect the **current time** hidden state, and consequently predicted output. The **weight matrices** handle

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

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.

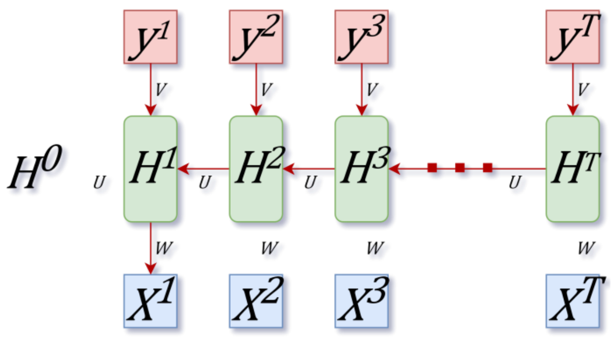
H^0 is assumed to be a null matrix.

Back propagation

Note: H, X and Y are assumed to be row vectors for our application. If they are instead saved as column vectors in computer memory, transpositions during backpropagation change accordingly.

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 states $H^{t-1} \cdots H^1$ have to be differentiated using chain rule:

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

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

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

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

$$\therefore \frac{\partial E}{\partial W} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \overbrace{\frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^2}{\partial H^1}}^{t\ times} \frac{\partial H^1}{\partial W} \qquad \frac{\partial E}{\partial U} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \frac{\partial \hat{y}^t}{\partial H^t} \overbrace{\frac{\partial H^t}{\partial H^{t-1}} \cdots \frac{\partial H^2}{\partial H^1}}^{t\ times} \frac{\partial H^1}{\partial U} \qquad (9)$$

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

Transpositions of matrix are taken as required, similar to ANN equation (5).

$\frac{\partial \varepsilon^t}{\partial \hat{y}^t} = \frac{\partial}{\partial \hat{y}} \left(\frac{1}{2} (y^t - \hat{y}^t)^2 \right) = (y^t - \hat{y}^t)$	$\frac{\partial \hat{y}^t}{\partial H^t} = (y^t - \hat{y}^t) \cdot (V)^{Tp}$	(10a)
$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$		

$$using\ quotient\ rule$$

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

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

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

$$= 1 - tanh^2(x) \tag{10b}$$

Using 10b: (11)

$$\frac{\partial \varepsilon^t}{\partial a^t} = \frac{\partial \varepsilon^t}{\partial \hat{y}^t}\frac{\partial \hat{y}^t}{\partial H^t}\frac{\partial H^t}{\partial a^t} = (1 - ((H^t))^2)\odot((y^t - \hat{y}^t)\cdot(V)^{Tp}) = \delta a^t$$

$$\frac{\partial \varepsilon^t}{\partial H^{t-1}} = \frac{\partial \varepsilon^t}{\partial \hat{y}^t}\frac{\partial \hat{y}^t}{\partial H^t}\frac{\partial H^t}{\partial a^t}\frac{\partial H^t}{\partial H^{t-1}} = \delta a^t \cdot (U)^{Tp} = \delta H^t$$

$$substituting\ (10a)\ and\ (11)\ \ into\ (9) \tag{12}$$

$$\frac{\partial E}{\partial V} = \sum_{t=0}^{T}(H^t)^{Tp}\cdot(y^t - \hat{y}^t)$$

$$\frac{\partial E}{\partial W} = \sum_{t=0}^{T}(X^t)^{Tp}\cdot\left(\overbrace{\delta H^T \odot \cdots \delta H^1}^{t\ times}\odot\delta a^1\right)$$

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{T}(H^{t-1})^{Tp}\cdot\left(\odot\overbrace{\delta H^T \odot \cdots \delta H^1}^{t\ times}\odot\delta a^1\right)$$

Update Weights

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

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

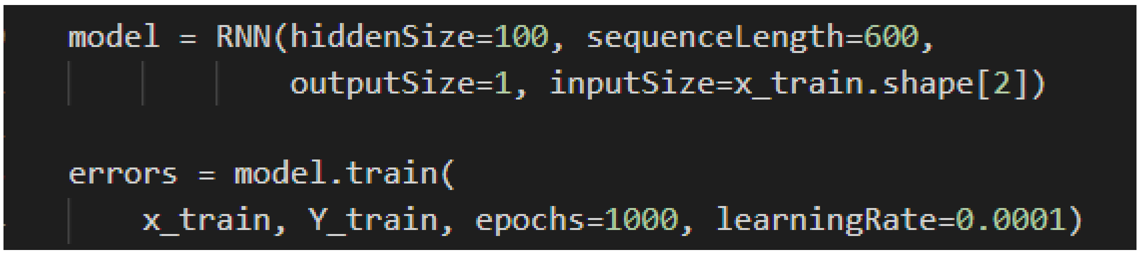
$$V(new) = \ V(old) - \lambda\odot\frac{\partial E}{\partial V}V(old)\ \ ;\ \ where\ \lambda = learning\ rate.$$

Application of RNN and Visualisations

It is important to apply a technique known as gradient clipping. This essentially means that if an element in the matrices and vectors that store the derivatives goes over or below a certain value, then it is clipped or reset back to that value. This is required as gradients are accumulated through all timesteps and can reach very large values, causing overflow errors, as well as general numerical instability.

Similar to before, 86 stock indicators for each day were collected and passed as input to the RNN with a hidden vector size of 100, and learning rate was set to 0.001, trained for 1000 epochs.

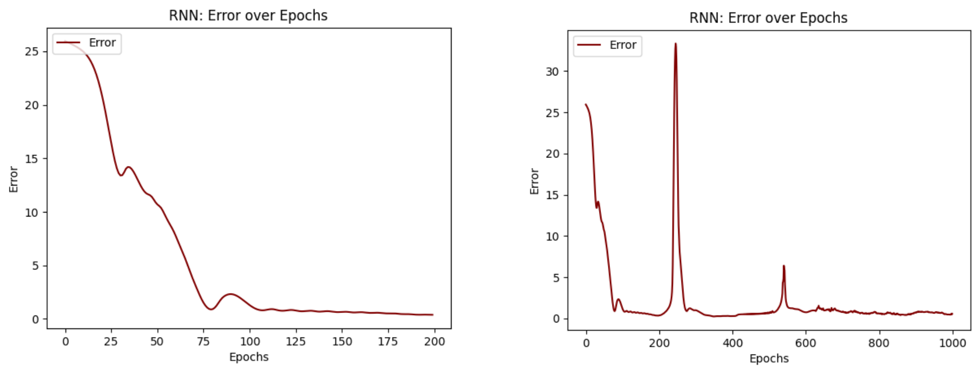
At the end, train dataset mean squared error loss was 0.0009505. Other tests were also done during different parameters. Two such tests are included.

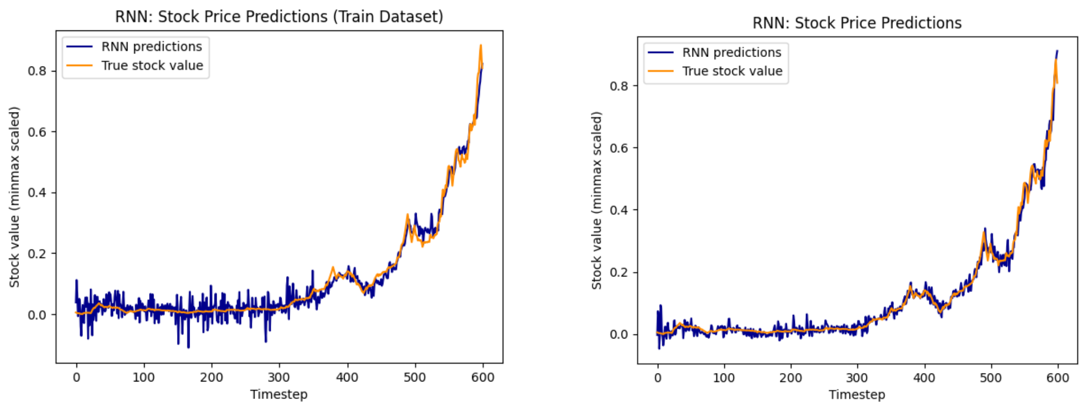


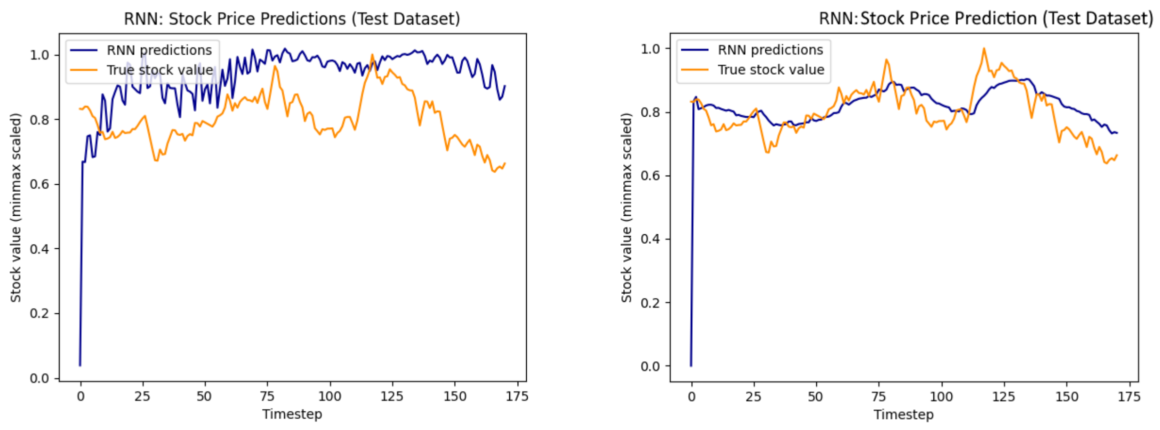
```
model = RNN(hiddenSize=100, sequenceLength=600,
            outputSize=1, inputSize=x_train.shape[2])

errors = model.train(
    x_train, Y_train, epochs=1000, learningRate=0.0001)
```

Figure 5: Snippet of instancing RNN model class. The RNN model class was coded by us from scratch with online code as reference





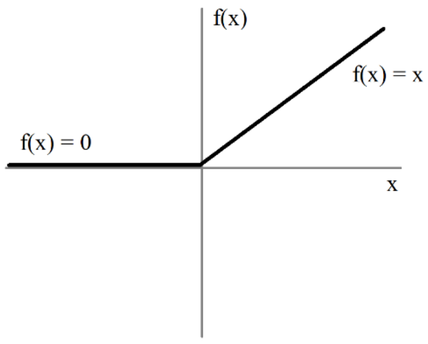


Vanishing gradient problem:

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 states are multiplied by numbers < 1 repeatedly (in forward propagation). The hidden states 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.

Solutions to Vanishing gradient problem:

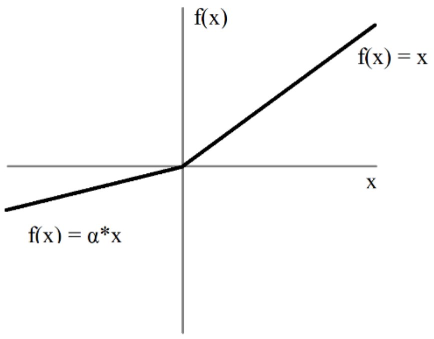


Another function used instead of sigmoid is defined as a hybrid function:

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

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.



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

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.

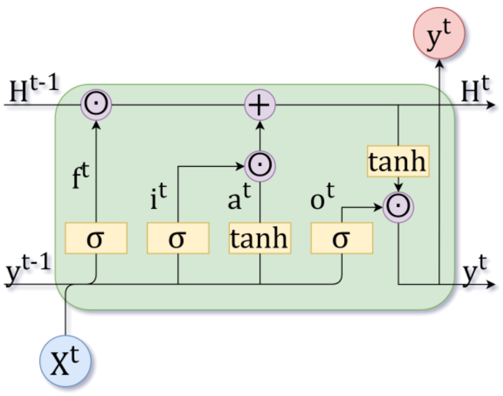
Section IV Long Short Term Memory (LSTM) Neural Networks (NN)

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.

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.

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 matrices of each layer.

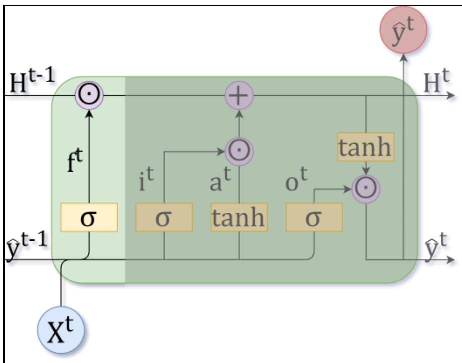
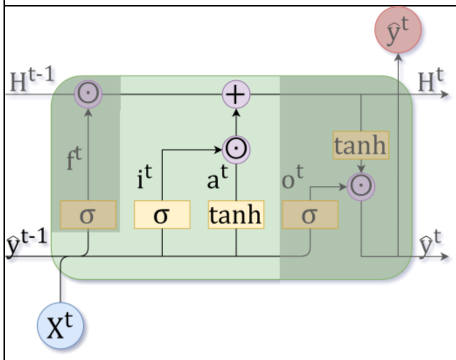
W_f, W_i, W_a, W_o are all recurrent weight matrices of each layer.

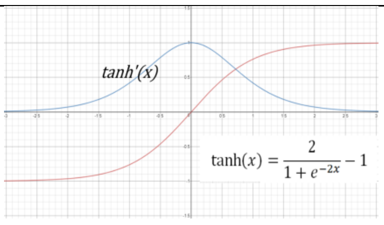
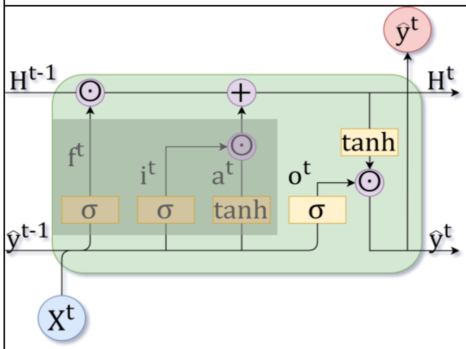
b_f, b_i, b_a, b_o are all scalar biases of each layer.

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:

 ŷ^t H^{t-1} H^t f^t i^t a^t o^t tanh σ σ tanh σ y^{t-1} ŷ^t X^t	"Forget layer"(f^t) decides what information is to be discard from the Hidden state. 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 state vector(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.
$f^t = (W_f \cdot X^t + U_f \cdot H^{t-1} + b_f)$ $H^t = \boldsymbol{f^t \odot H^{t-1}} + a^t \odot i^t$	
 ŷ^t H^{t-1} H^t f^t i^t a^t o^t tanh σ σ tanh σ y^{t-1} ŷ^t X^t	"Input layer"($a^t \odot i^t$) decides to what extent the current time input and previous time output update the Hidden state vector with new information. $a^t \odot i^t$ is **added** to form the Hidden state vector, 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.

$i^t = (W_i \cdot X^t + U_i \cdot H^{t-1} + b_i)$ $a^t = (W_a \cdot X^t + U_a \cdot H^{t-1} + b_a)$ $H^t = f^t \odot H^{t-1} + \boldsymbol{a^t} \odot \boldsymbol{i^t}$	 In stock price prediction, this layer processes how the new stock price information is added to the Hidden state vector.
 y^t H^{t-1} H^t f^t i^t a^t o^t tanh σ σ tanh σ y^{t-1} y^t X^t	The filtered information from above set of operations is the used to calculate outputs. The Hidden state vector is processed through tanh and then scaled by the output layer to calculate the predicted outputs. $o^t = (W_o \cdot X^t + U_o \cdot H^{t-1} + b_o)$ $H^t = f^t \odot H^{t-1} + \boldsymbol{a^t} \odot \boldsymbol{i^t}$ $\hat{y}^t = \tanh(H^t) \odot o^t$

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 $\varepsilon^t = \frac{1}{2}(y^t - \hat{y}^t)^2$

$\frac{\partial E}{\partial W_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t\ times} \frac{\partial \hat{y}^1}{\partial W_\theta}$	$\frac{\partial E}{\partial U_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t\ times} \frac{\partial \hat{y}^1}{\partial U_\theta}$	$\frac{\partial E}{\partial b_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t\ times} \frac{\partial \hat{y}^1}{\partial b_\theta}$	(13)
$\frac{\partial E}{\partial b_\theta} = \sum_{t=0}^{T} \frac{\partial \varepsilon^t}{\partial \hat{y}^t} \overbrace{\frac{\partial \hat{y}^t}{\partial \hat{y}^{t-1}} \frac{\partial \hat{y}^{t-1}}{\partial \hat{y}^{t-2}} \cdots \frac{\partial \hat{y}^2}{\partial \hat{y}^1}}^{t\ times} \frac{\partial \hat{y}^1}{\partial b_\theta}$			
$\frac{\partial \varepsilon^t}{\partial \hat{y}^t} = \frac{\partial}{\partial \hat{y}}\left(\frac{1}{2}(y^t - \hat{y}^t)^2\right) = (y^t - \hat{y}^t)$			(14)
$tanh(x) = \frac{2}{1+e^{-2x}} - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ $using\ quotient\ rule$ $tanh'(x) = \frac{-(e^x - e^{-x})(e^x - e^{-x}) + (e^x + e^{-x})(e^x + e^{-x})}{(e^x + e^{-x})^2}$			

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

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

$$= 1 - tanh^{2}(x) \tag{15}$$

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

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

$$= (U_{O})^{Tp} \cdot ((o^{t}) \odot (1-(o^{t})))$$

$$+ (U_{i})^{Tp} \cdot o^{t} \odot (1 - tanh^{2}(H^{t})) \odot a^{t} \odot ((i^{t}) \odot (1-(i^{t})))$$

$$+ (U_{a})^{Tp} \cdot o^{t} \odot (1 - tanh^{2}(H^{t})) \odot i^{t} \odot (1-((a^{t})^{2}))$$

$$+ (U_{f})^{Tp} \cdot o^{t} \odot \left(1 - tanh^{2}(H^{t})\right) \odot H^{t-1} \odot ((f^{t}) \odot (1-(f^{t})))$$

$$= \delta \hat{y}^{t} \tag{16}$$

$$\frac{\partial \hat{y}^{t}}{\partial W_{O}} = \frac{\partial \hat{y}^{t}}{\partial o^{t}}\frac{\partial o^{t}}{\partial W_{O}} = tanh(H^{t}) \odot ((o^{t}) \odot (1-(o^{t}))) \otimes X^{t} = \delta W_{O}$$

$$\frac{\partial \hat{y}^{t}}{\partial W_{i}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial i^{t}}\frac{\partial i^{t}}{\partial W_{i}} = o^{t} \odot (1 - tanh^{2}(H^{t})) \odot a^{t} \odot ((i^{t}) \odot (1-(i^{t}))) \otimes X^{t} = \delta W_{i}$$

$$\frac{\partial \hat{y}^{t}}{\partial W_{a}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial a^{t}}\frac{\partial a^{t}}{\partial W_{a}} = o^{t} \odot (1 - tanh^{2}(H^{t})) \odot i^{t} \odot (1-((a^{t})^{2})) \otimes X^{t} = \delta W_{a}$$

$$\frac{\partial \hat{y}^{t}}{\partial W_{f}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial f^{t}}\frac{\partial f^{t}}{\partial W_{f}} = o^{t} \odot \left(1 - tanh^{2}(H^{t})\right) \odot H^{t-1} \odot ((f^{t}) \odot (1-(f^{t}))) \otimes X^{t} = \delta W_{f} \tag{17}$$

$$\frac{\partial \hat{y}^{t}}{\partial U_{O}} = \frac{\partial \hat{y}^{t}}{\partial o^{t}}\frac{\partial o^{t}}{\partial U_{O}} = tanh(H^{t}) \odot ((o^{t}) \odot (1-(o^{t}))) \otimes \hat{y}^{t-1} = \delta U_{O}$$

$$\frac{\partial \hat{y}^{t}}{\partial U_{i}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial i^{t}}\frac{\partial i^{t}}{\partial U_{i}} = o^{t} \odot (1 - tanh^{2}(H^{t})) \odot a^{t} \odot ((i^{t}) \odot (1-(i^{t}))) \otimes \hat{y}^{t-1} = \delta U_{i}$$

$$\frac{\partial \hat{y}^{t}}{\partial U_{a}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial a^{t}}\frac{\partial a^{t}}{\partial U_{a}} = o^{t} \odot (1 - tanh^{2}(H^{t})) \odot i^{t} \odot (1-((a^{t})^{2})) \otimes \hat{y}^{t-1} = \delta U_{a}$$

$$\frac{\partial \hat{y}^{t}}{\partial U_{f}} = \frac{\partial \hat{y}^{t}}{\partial H^{t}}\frac{\partial H^{t}}{\partial f^{t}}\frac{\partial f^{t}}{\partial U_{f}} = o^{t} \odot \left(1 - tanh^{2}(H^{t})\right) \odot H^{t-1} \odot ((f^{t}) \odot (1-(f^{t}))) \otimes \hat{y}^{t-1} = \delta U_{f} \tag{18}$$

$substituting\ (14), (16), and\ (17)\ into\ (13)$	$\frac{\partial E}{\partial W_\theta} = \sum_{t=0}^{T} (y^t - \hat{y}^t) \overbrace{\odot\, \delta\hat{y}^t \cdots \odot\, \delta\hat{y}^1}^{t\ times} \odot\, \delta W_\theta$	
$substituting\ (14), (16), and\ (18)\ into\ (13)$	$\frac{\partial E}{\partial U_\theta} = \sum_{t=0}^{T} (y^t - \hat{y}^t) \overbrace{\odot\, \delta\hat{y}^t \cdots \odot\, \delta\hat{y}^1}^{t\ times} \odot\, \delta U_\theta$	(19)
$substituting\ (14), and\ (16)\ into\ (13)$	$\frac{\partial E}{\partial b_\theta} = \sum_{t=0}^{T} (y^t - \hat{y}^t) \overbrace{\odot\, \delta\hat{y}^t \cdots \odot\, \delta\hat{y}^1}^{t\ times}$	

Update Weights

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

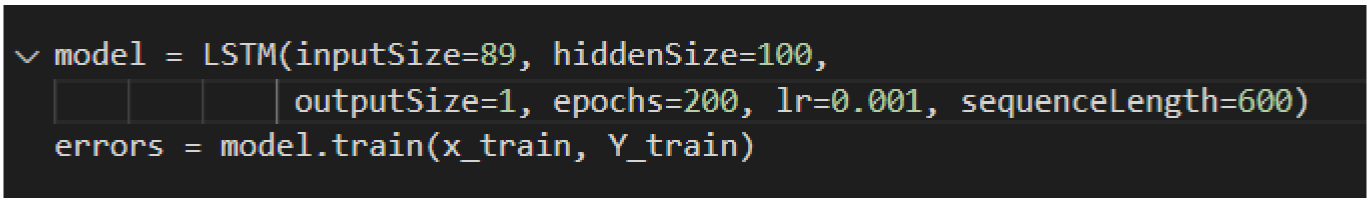
$$b_\theta(new) = b_\theta(old) - \lambda \odot \frac{\partial E}{\partial b_\theta} b_\theta(old)\ ; \quad where\ \lambda = learning\ rate.$$

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

Application of LSTM neural networks

Similar to RNN, the LSTM model was given 86 stock market indicators for each day. Hidden vector size was set to 100, and learning rate was set to 0.001. Trained for 200 epochs. Gradient clipping was set to maximum 5 and minimum -5.

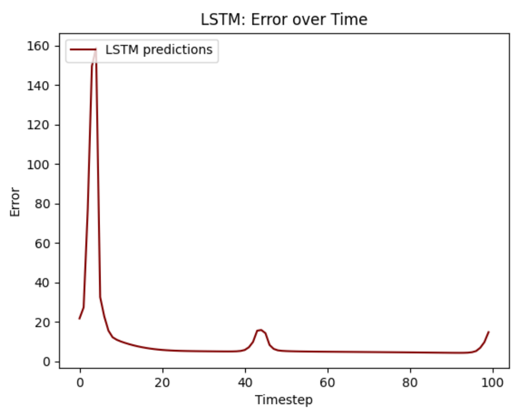
Final mean squared error loss value was 0.0003694. Other tests were also done during different parameters. Two such tests are included.

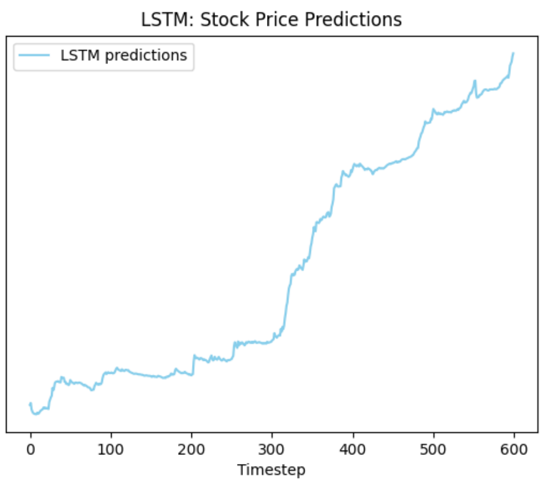


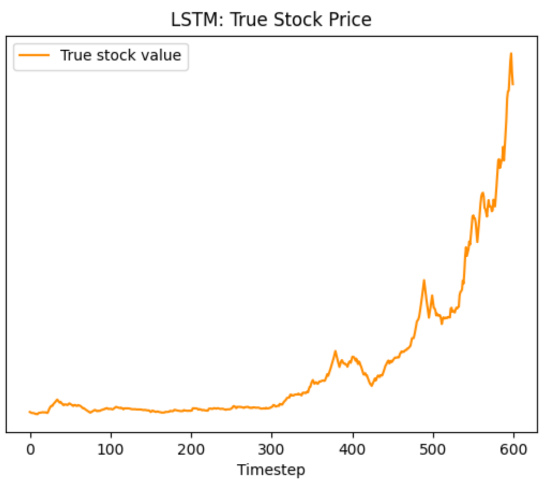
```
model = LSTM(inputSize=89, hiddenSize=100,
             outputSize=1, epochs=200, lr=0.001, sequenceLength=600)
errors = model.train(x_train, Y_train)
```

Figure 6: Code snippet of instancing the LSTM model class. The LSTM model class was coded by us from scratch with online code as reference

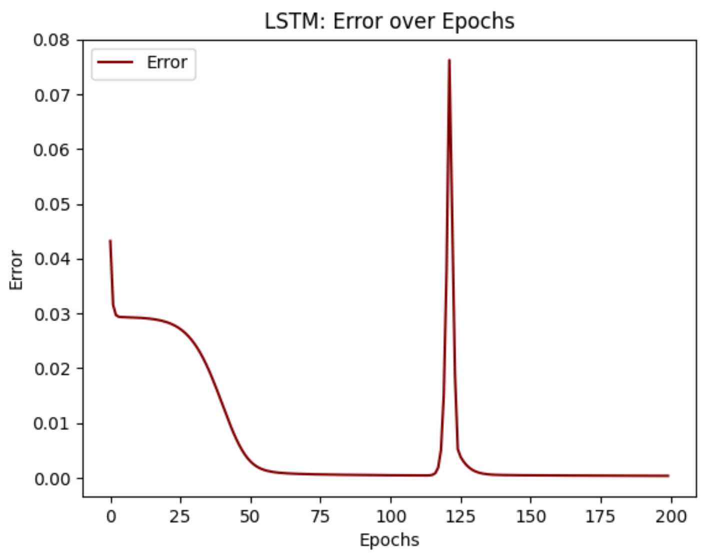
Test 1:

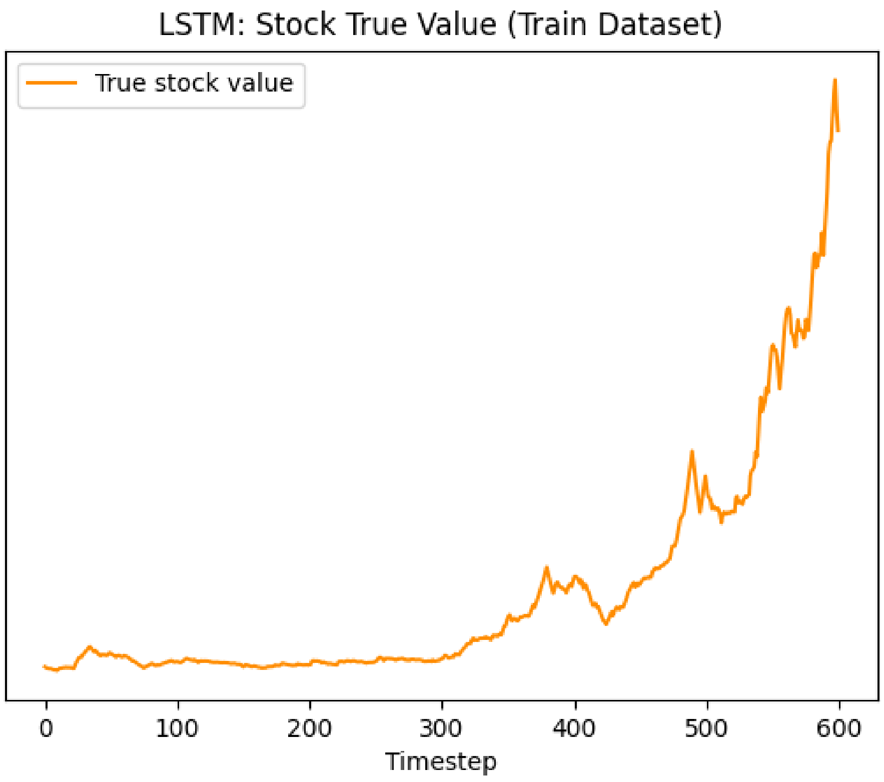

LSTM: Error over Time
LSTM predictions
Error
Timestep
0
20
40
60
80
100
120
140
160

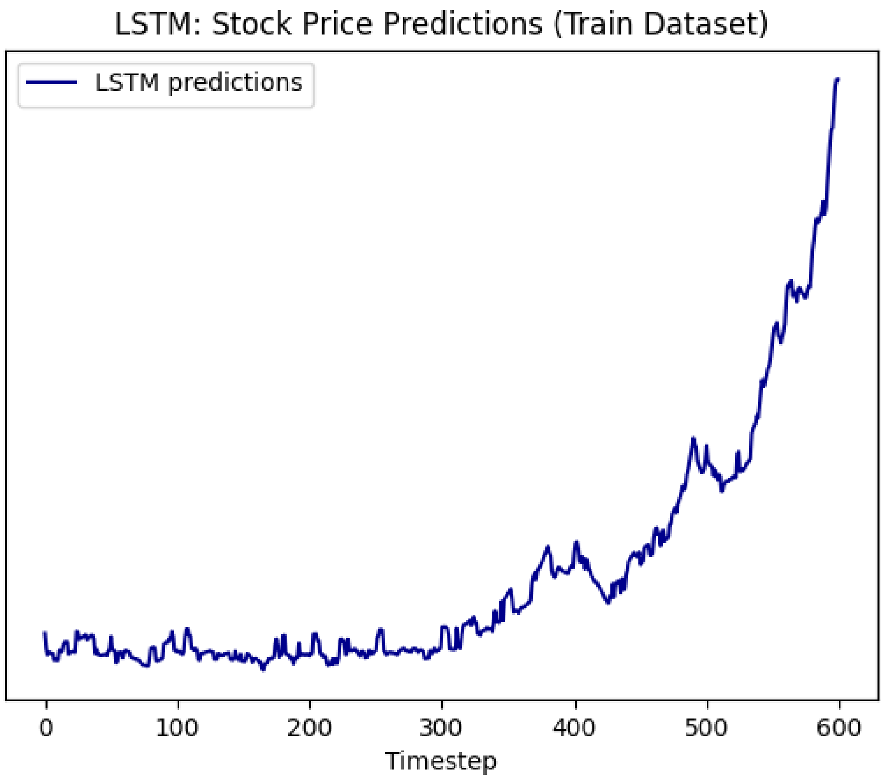

LSTM: Stock Price Predictions
LSTM predictions
Timestep
0
100
200
300
400
500
600

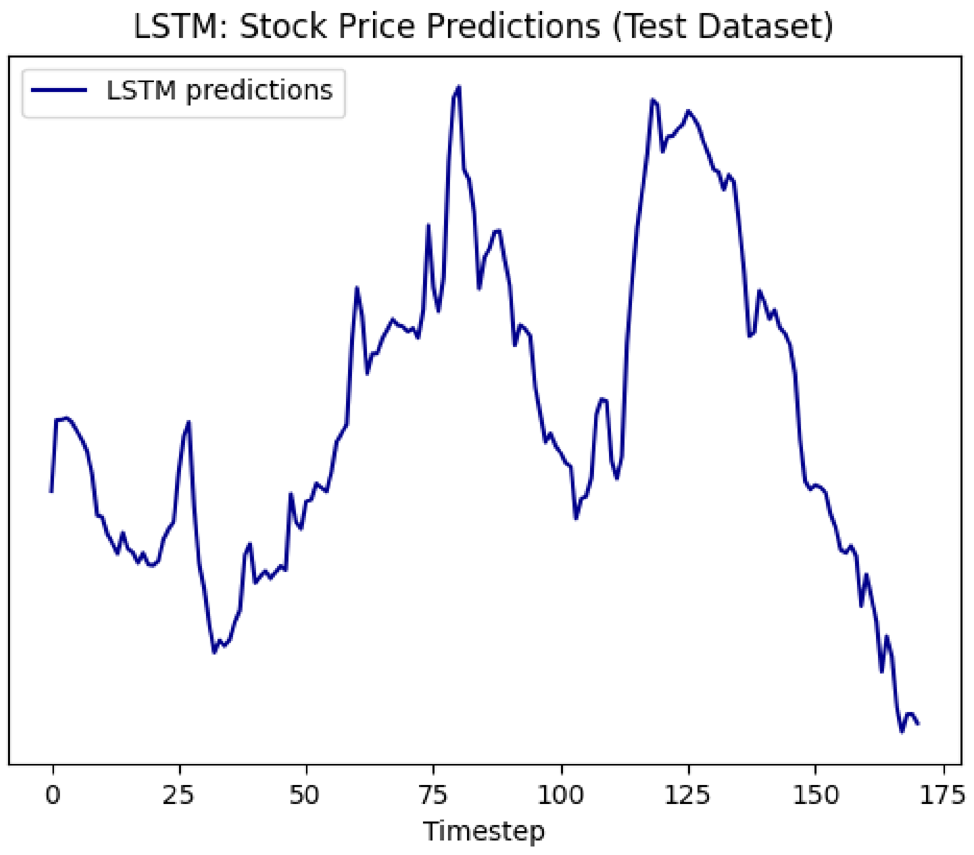

LSTM: True Stock Price
True stock value
Timestep
0
100
200
300
400
500
600

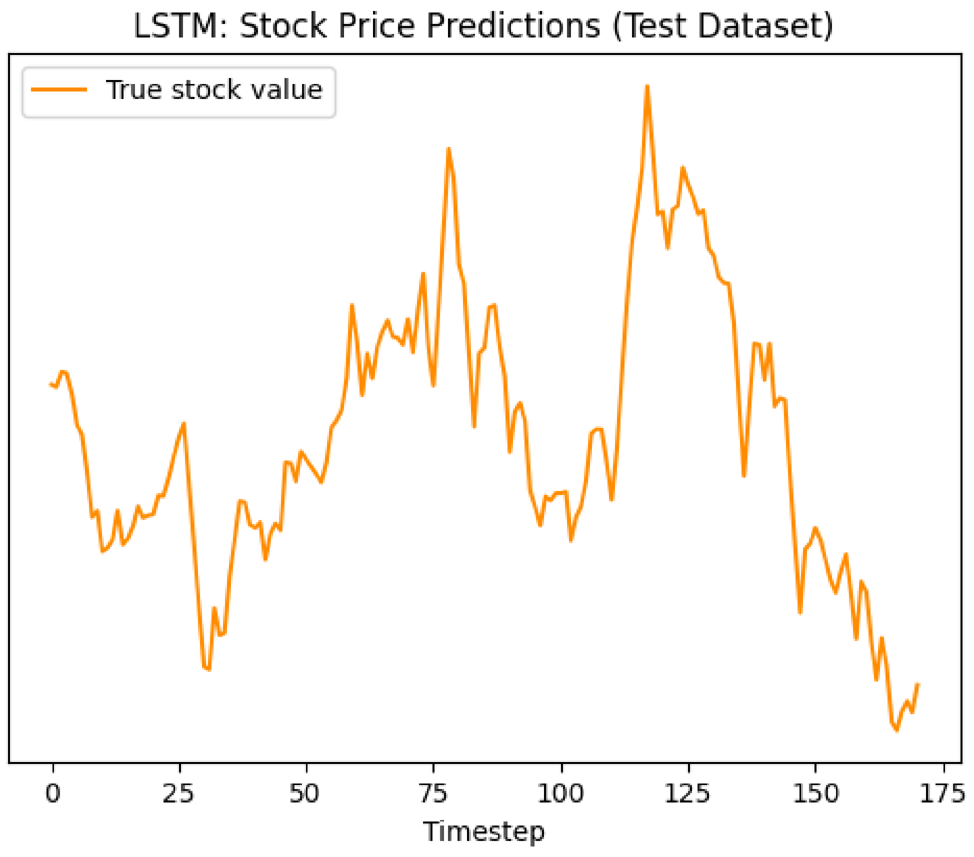
Test 2:


LSTM: Error over Epochs
Error
Error
Epochs
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
25
50
75
100
125
150
175
200


LSTM: Stock True Value (Train Dataset)
True stock value
0
100
200
300
400
500
600
Timestep


LSTM: Stock Price Predictions (Train Dataset)
LSTM predictions
0
100
200
300
400
500
600
Timestep


LSTM: Stock Price Predictions (Test Dataset)
LSTM predictions
0
25
50
75
100
125
150
175
Timestep


LSTM: Stock Price Predictions (Test Dataset)
True stock value
0
25
50
75
100
125
150
175
Timestep

Innovation

Only the next day's closing stock price was predicted up till now, however, often, longer term predictions are required to make educated decisions about trading and investments in stocks. Furthermore, mean squared error was used as we were mostly concerned with predicting stocks and not generating profits from trading/investment. However, mean squared error does not give us much of an understanding of how sure the network itself is about its predictions. If we simply invested a constant number of stocks based on LSTM/RNN predictions, we would quickly go bankrupt (as tested by us during the project). Mean squared error when drawing just the next day's sample also does not give an accurate picture of the medium term trends which is required for investing, trading, or portfolio balancing.

Our solution:

Custom loss function:

Current libraries can incorporate custom loss functions by using automatic differentiation. Thus, if make an LSTM model that returns predictions $\hat{y}^t$ as the amount of stocks to sell/buy on any particular day, (negative value for sell, positive value for buy), then we can make an automatic trading bot that learns to make stock trading decisions on its own in order to maximise profits over a particular trading period.

```
def customLoss(y_pred, y_true):
    # y_pred should be a list of numbers. Each representing how many shares to buy/sell
each day
    # negative values for selling, positive for buying.
    capital = 100-K.sum(y_true * y_pred)
    # sell all residual shares to get capital on last day
    capital += (K.sum(y_pred) * y_true[..., -1])
    ROI = ((capital - 100) / 100)
    return ROI * -1
```

Since we are returning a loss value to the model and minimizing that, the ROI is multiplied by -1 (flipped).

Epoch 200/200

Train loss: -1.4383. Train ROI: 143.83%

Test loss: -0.1103. Train ROI: 11.03%

However, this loss function does not put typical constraints that capital and stock markets would face. Such as having limited capital, and not being able to short markets for more than intraday trading as per Indian stock market laws.

This customLoss function tries to incorporate some of those limitations:

```
def customLoss(y_pred, y_true):
    # y_pred should be a list of numbers. Each representing how many shares to buy/sell
each day
    # negative values for selling, positive for buying.
    capital = 100000.0
    heldShares = 0.0
    for i in range(0, K.int_shape(y_pred)[1]):
        # can only buy shares if capital > 0
        if (capital > 0 and y_pred[..., i] > 0):
            capital -= y_true[..., i] * y_pred[..., i]
            heldShares += 1
        # can only sell shares if shares are already owned
        if (y_pred[..., i] < 0 and heldShares > 0):
            capital -= y_true[..., i] * y_pred[..., i]
            heldShares -= y_pred[..., i]
    # sell all residual shares to get capital on last day
    capital += heldShares * y_true[..., -1]
    ROI = ((capital - 100000) / 100000)
    return ROI * -1
```

Furthermore, when investing in multiple stocks, the custom loss function could be modified to invest a fixed capital for portfolio balancing and management and maximize investment returns over a large stock portfolio.

Limitations

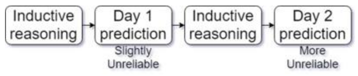
The predictions obtained may not work as well in the future.

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 about 600 to 750 days, 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.



To optimise the model further, we would want to train the model on a large dataset of multiple stocks and then finetune it for a particular stock with a lower learning rate. This is known as transfer learning.

Libraries that implement such models have several optimisations nowadays such as:

1. Adaptive momentum learning rate optimisation: is a learning rate technique that optimises the learning rate for each batch and iteration of the model instead of having a constant learning rate.
2. Dropouts: is a technique where a fixed percentage of the entries in the weight and bias matrices are randomly reinitialised to 0. This causes the weights of the neural network in the columns and rows of the dropped-out element to adapt to the increased loss, thus increasing regularisation and promoting denser matrices. This also reduces overfitting of the model to the training data.
3. Early stopping: Models are often stopped early in order to reduce overfitting on the training dataset.
4. Reduce learning rate on plateau: Model reduces learning rate when it stops learning and fluctuates back and forth. This reduces fluctuations and lets the model fine-tune down to the minimum value.
5. GPU acceleration: using GPUs allow for parallelisation. GPUs also have higher bandwidth memory and tensor cores which are optimised for deep learning tasks, thus improving performance by several magnitudes.

However, all of this requires very complicated coding, some of which, like CUDA, we haven't even learned yet. Furthermore, we have already spent hundreds of hours understanding, coding, report writing, the project is already immensely complex as it is. Adding these other features does not seem feasible as it would increase complexity even further and require several hundred more hours to code it from scratch.

The report works just as an exercise to understand the math and the computations behind the model better, and understand why/how these models actually work.

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.

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 plots, and the final mean squared error loss value, LSTMs seem to be the best option for stock price prediction.

Our innovative strategy of Investments based on the LSTM with the custom loss function, when compared to safer investments like fixed deposit, government bonds, and mutual funds, which usually have 6% to 12% mean ROI per year[21] was not more effective in generating returns on investment at ~11%. However, if this model was further improved by using a library-based solution to address the limitations discussed then the ROI could improve. Furthermore, having multiple LSTM models for each of NSE India's 500+ stocks would increase the ROI as the model could exploit chances of earning money more often. 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.

Almost all sequential data analysis and tasks have been moved almost completely to Machine Learning models so it is good to explore the mathematics and computations that such models use in varied applications.

Our Learnings:

This report has given us a greater appreciation for the applications of mathematics, especially Linear Algebra. I have found out that Linear Algebra, through neural networks, can be applied to any field of studies, including commerce.

[21] *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: Visual Studio Code, Microsoft Excel, google colabatory, python, python libraries such as numpy, pandas, matplotlib, 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, 2021.

"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, 2021.

"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, 2021.

"Definition of Outer Product", *Chegg Study*, 2018. https://www.chegg.com/homework-help/definitions/outer-product-33 Accessed on: 7August, 2021.

"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, 2021.

"How to Multiply Matrices," *Advanced Math is Fun*, 2017, https://www.mathsisfun.com/algebra/matrix-multiplying.html Accessed on: 4 August, 2021.

Indian Mutual Funds Handbook: A Guide for Industry Professionals and Intelligent Investors, by Sundar Sankaran, Vision Books Pvt. Ltd., 2003, Accessed on 17 September, 2021.

"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, 2021.

"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, 2021.

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, 2021

"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, 2021.

"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, 2021.

"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, 2021.

"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, 2021.

"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, 2021.

"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, 2021.

View all posts by christinakouridi. (2019, June 20). *Implementing a LSTM from scratch with Numpy*. Christina's Blog; Christina's blog. https://christinakouridi.blog/2019/06/20/vanilla-lstm-numpy/

Aditya Singh. (2021, May 22). *Implementing A Recurrent Neural Network (RNN) From Scratch*. Analytics India Magazine. https://analyticsindiamag.com/implementing-a-recurrent-neural-network-rnn-from-scratch/

Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(1), 1–42. https://doi.org/10.1162/neco.1997.9.1.1

Ross, Sheldon M. Stochastic Processes. (2011). John Wiley & Sons, 1995. https://books.google.co.in/books/about/Stochastic_Processes.html?id=qiLdCQAAQBAJ&redir_esc=y

"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, 2021.

Aidan Gomez. (2016, April 18). *Backpropogating an LSTM: A Numerical Example - Aidan Gomez - Medium*. Medium; Medium. https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9

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.

Bounthavong, M. (2018, March 15). Mark Bounthavong. Mark Bounthavong. https://mbounthavong.com/blog/2018/3/15/generating-survival-curves-from-study-data-an-application-for-markov-models-part-2-of-2

cheillanju. (2021). *ann-from-scratch/ann-from-scratch.ipynb at master · cheillanju/ann-from-scratch*. GitHub. https://github.com/cheillanju/ann-from-scratch/blob/master/ann-from-scratch.ipynb

Zedric Cheung. (2020, June 20). *Customize loss function to make LSTM model more applicable in stock price prediction*. Medium; Towards Data Science. https://towardsdatascience.com/customize-loss-function-to-make-lstm-model-more-applicable-in-stock-price-prediction-b1c50e50b16c

Yugesh Verma. (2021, October 16). *A Guide to Hidden Markov Model and its Applications in NLP*. Analytics India Magazine. https://analyticsindiamag.com/a-guide-to-hidden-markov-model-and-its-applications-in-nlp/

Pykes, K. (2020, November 25). *Part Of Speech Tagging for Beginners - Towards Data Science*. Medium; Towards Data Science. https://towardsdatascience.com/part-of-speech-tagging-for-beginners-3a0754b2ebba

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.

Brandt, K., Michael, P., & Pedersen, S. (n.d.). *The Matrix Cookbook*. https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

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

Ryan, M. (2020, June 6). *A Very Simple Method of Weather Forecast Using Markov Model Lookup Table*. Medium; Towards Data Science. https://towardsdatascience.com/a-very-simple-method-of-weather-forecast-using-markov-model-lookup-table-f9238e110938

Brilliant Math & Science Wiki. (2021). Brilliant.org. https://brilliant.org/wiki/markov-chains/

Chegg.com. (2021). Chegg.com. https://www.chegg.com/homework-help/definitions/outer-product-33

Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(1), 1–42. https://doi.org/10.1162/neco.1997.9.1.1

Ross, Sheldon M. Stochastic Processes. (2011). John Wiley & Sons, 1995. https://books.google.co.in/books/about/Stochastic_Processes.html?id=qiLdCQAAQBAJ&redir_esc=y

"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, 2021.

Aidan Gomez. (2016, April 18). *Backpropogating an LSTM: A Numerical Example - Aidan Gomez - Medium*. Medium; Medium. https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9

Ruineihart, D., & Williams, R. (n.d.). LEARNING INTERNAL REPRESENTATIONS BERROR PROPAGATION two. https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Flat Minima. *Neural Computation*, *9*(1), 1–42. https://doi.org/10.1162/neco.1997.9.1.1

Bounthavong, M. (2018, March 15). Mark Bounthavong. Mark Bounthavong. https://mbounthavong.com/blog/2018/3/15/generating-survival-curves-from-study-data-an-application-for-markov-models-part-2-of-2