An in Depth Analysis of Neural Network with Application in Finance

by

Noshin Tasnim
15301004
Farhana Yasmeen
15301024

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
April 2019

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Noshin Tasnim	Farhana Yasmeen
15301004	15301024

Approval

The thesis/project titled "An in Depth Analysis of Neural Network with Application in Finance" submitted by

1. Noshin Tasnim (15301004)
2. Farhana Yasmeen (15301024)

Of Spring, 2019 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on April 25, 2019.

Examining Committee:

Supervisor:
(Member)

Mahbub Alam Majumdar
Professor
Department of Computer Science and Engineering
BRAC University

Program Coordinator:
(Member)

Jia Uddin
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Md. Abdul Mottalib
Professor
Department of Computer Science and Engineering
Brac University

Abstract

Artificial neural network model is inspired from how the nervous system of brain works. If it is designed properly it can process large amount of data or information and can give proper output for different application like pattern recognition, forecasting disease or financial data etc. In recent years Artificial neural network has been a great choice to analyze financial time series data as they are quite capable of learning the relationships among different features of data. As the world's economy is continuously changing, there is a need for keep an eye on the dynamic conditions of economy. Therefore, financial institutions and investors always wants a reliable system to monitor the data relationship so that they can simulate and predict financial positions on the basis of market trends in order to find where should they invest. But because of the high volatility and high non linearity it has been quite a challenge to predict the financial stock market.

In this paper we attempt to study neural network and how they are actually useful in predicting stock market and finally we are going to use different model of ANN to predict the stock price of Amazon and SP 500 index. We will analyze the capability of neural net to cope with the nonlinear and chaotic patterns of data and their ability to predict.

Keywords: Neural Network, Stock Market, Prediction, Deep layer, Hidden layer

Acknowledgement

This is the work of Noshin Tasnim and Farhana Yasmeen- students of the CSE department of BRAC University. The document has been prepared as an effort to organize the knowledge acquired by us about the thesis. All thanks to Almighty who provided us guidance and capabilities to complete this research. We would like to express our sincere gratitude to our supervisor for his invaluable guidance, comments and suggestions during the course of our thesis. The completion of this study could not have been possible without his continuous support regarding the study. We are also grateful to the faculty members of the department of Computer Science and Engineering of BRAC University, who have always been helpful and kind towards us in this whole study period. Last but not the least we would like to thank our parents, family and friends who constantly give us support and encouragement.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

In recent days, one of the buzzing topics is neural network. It's tremendous success in field of science has made it so popular. Use of neural network gives more efficient and faster result. It is used for prediction, classification, regression, pattern recognition etc. For its excellent performance, it has now spread in every sector such as finance, medical diagnosis, robotics, language recognition, autonomous vehicle, computer vision etc. The reason for the high success rate and good accuracy is it can learn complex non-linear representation of input data. Now a days, our goal is to lead a comfortable, easy life. Neural network's technology is used as a substitute of human. It has made life much easier as it can perform work more accurately and faster than a human. In the near future, work of neural network will bring an extraordinary change in the field of science. The main attraction of neural network is that its structure is similar to human brain. As it is closed to the structure of neurons, it is known as artificial neural network.

In our brain, there are millions of neurons. These neurons are connected to each other like a web. A neuron processes an information and passes that processed information from one neuron to another neuron. Neurons get information from environment and process it to make a decision. Neural network is similar to biological neurons structure and work. In neural network, there are a lot of neurons. They are connected with each other. These neurons are arranged in different layers, each layer contains a number of neurons. There are three types of layers in neural network-input layer, output layer and hidden layer. There can be multiple hidden layer. A neural network with lots of hidden layer is called deep learning. Input layer takes the input and pass it to hidden layer. The hidden layers process it and pass it from one hidden layer to another hidden layer. After all the processing, the result is shown in the output layer. This is how neural network works like human brain.

As the use of neural network is increasing, many works have already done in this area to make neural network better. More work should be done to make neural network more efficient, optimized and faster. in this paper we have talked about how a neural network works, how can we train a neural network, how we can make the computation more efficient and faster, what optimization should be implemented on various methods of neural network. Our work will help to make neural network

methods better to implement them on various application of everyday life.

There are so many applications of neural network these days. In this paper, we will talk about its application in finance. As neural network is very good in forecasting, we will show the application in finance by prediction of stock market. A stock market is a market where ownership exchange of stocks, shares, government bonds take place. In finance, stock market plays a huge role not only on a nation's economy but also in world's economy. Investors, business people invest their money on stock market and try to anticipate it by illustrating people's opinion, economic condition, political condition. People want to make their profit larger along with having minimum risk. For that, a good prediction of stock market is very necessary. Stock market prediction is one of the toughest jobs as the market is very erratic and unpredictable. It is very had to get 100% accuracy for this prediction. In last decade, many research works have taken place trying new methods, techniques to predict stock market. But none of them was successful enough to get 100% accuracy. That is why the research is still going on.

In this paper, we have used neural network methods to predict stock market. As it the most powerful tool for forecasting. As neural network is good with time series, it handles stock market prediction very well. It works very well to recognize complex pattern, make a connection between variables and handle complex data like stock market. So, neural networks method will work best for predicting stock market. Our work will help the investors and business people to have a great overview of the stock market price so that they can make proper decision and make profit with minimum risk.

For prediction of stock market, we have used data of S&P 500 which is an American stock market index on 500 large companies' market capitalization. And also, we used stock market data of one of the biggest companies in the world, Amazon. We used data from 2016-2019 to predict the stock market. We have applied several methods of neural network to predict the stock market for both data sets. Then we have analyzed them to see how well our applied method works to give good prediction of the stock market.

1.2 Motivation

There are many popular methods to perform classification, regression and forecasting. But neural network performs better than any other methods. Neural network has some amazing properties which make it work better for any complex data set or pattern recognition. Neural networks back propagation is very effective give a nearly accurate result. It works in parallel architecture which helps to reduce processing time which leads to training a huge data set in a short time. It can also build a non-linear relationship among the variables. These properties have made neural network attractive and so people are leaning towards it to implement in different areas.

Now a days, area where neural network has made a successful result is increasing. For predicting stock market, there has been a lot of researches. But, no significant

guidelines has been found to estimate or predict the market. Predicting the market is a challenge for the traders. Therefore, we have worked on predicting stock market using neural network as the performance of neural network is better than any other methods.

Considering the interesting features and hyped popularity, we have been motivated to work on neural network and its implementation in predicting stock market in this paper.

1.3 Literature Review

For several years researchers have put their interest in neural network model and how to use that model to solve different real life problem.

Ongoing research and proof demonstrate that financial market is nonlinear. Though in a research of fang (1994) their team shows significant result using linear method to analyze nonlinear financial structure [1]. But we should keep in mind that financial market is always evolving and with the increasing amount of data it will be quite difficult to understand the relationship in the data. In 1986 Bollerslev showed in his research that non linearity is exist in the financial market and created a model GARCH to determines the feature of time series.[2]

Later on ANN has emerged as a powerful model which can perform task like pattern recognition, classification, prediction etc. Lawrence (1991)[3], Minsky(1969)[4], nelson[1991] [5] shows elaborate study on neural network on how they work and perceptrons.

ANN can make a sense of nonlinear financial function because nn model is nonlinear data driven. Without any prior knowledge of data they can make connections in them which makes it highly applicable in financial data. Wong and Selvi (1998)[6]collected and provide survey about the applications in neural network in business by classifying the different article on the basis of publication year , area of application etc and showed their development and contribution. Chatterjee et al. (2000) [7] in his paper give a broad overview on using ANN in financial marketing. They have discussed different methodology of ANN and how it is better than other methodologies and also gives the successful implementation of NN in various institutions. Edward Gately has described some method about how to build train and test set using commercially available software for finical forecasting in his book "Neural network for financial forecasting". Zhang et al reviewed articles that tended to demonstrating issues when ANN is used for forecasting. Data types , size of training and test set, model architecture meaning number of layers and number of nodes in each layer, activation function, training algorithm, normalization or regularization methods etc are discussed as modelling issue in their paper.[8]

Leaonardo and Rosangela in their paper, they analyzed and examined the prediction of neural network to predict future trend of North American, European and Brazilian Stock Markets Indexes. They also applied GARCH to evaluate neural network accuracy. Mumtaz et al (2011) did a study on Turkish banks failure using artificial

neural network. Financial ratio which is indicator of the performance of bank was given, on that basis they use two model which are logistic model and ANN. They found out ANN model is better estimator about financial failure of bank than the logistic regression.[9] Fadlalla and Lin studied the recent trade of neural net in finance and addressed the features of these application and give a comparison with other economical statistical methods. [10]

Anyaeche and Ighravwe, (2013) used linear regression and neural network to measure the performance of prediction. They create a model of artificial neural network and back propagation artificial neural network as a predictive tool. They wanted to develop an interrelationship among productivity, price recovery and profitability to predict the performance. Here productivity and price recovery was independent variable on the other hand profitability was dependent variable. With these they proposed a 2-20-20-1 neural network architecture. In that study mean square error (MSE) of neural net architecture was 0.02 where multiple linear regression had a MSE of 0.036. They concluded their study by stating that ANN is better tool for making interrelationship among variables. [11]

Deep learning opens the door for many algorithm to shine with increased data. Such as convolutional neural network (CNN) is one type of deep learning algorithm which has shown its importance by giving surprisingly good result in different type of problem from time series to image classifications. A team of Tianyi Liu, Shuangsang Fang, Yuehui Zhao, Peng Wang, Jun Zhang gave a detailed analysis of the process of CNN algorithm both the forward process and back propagation. Using java they applied convolutional model to face detection problem. Theoretically they compute the efficiency by measuring actual time of forward and backward computation.[12]

Chapter 2

Structure of Neural Network

Neural Networks exist since at least 1958, yet only recently have it achieved outstanding performances in a wide range of large-scale.[13] It performs difficult tasks for machines, like classification, regression of huge data sets. Neural network is an artificial model of biological neurons. an interconnection of artificial neurons. It learns the relations between selected inputs and outputs from the previous experiences. The neurons receive inputs from the environment or other artificial neurons, then after adding weight transfer function transform the result into the output. The transfer function can be sigmoid, hyperbolic tangent functions or a step.

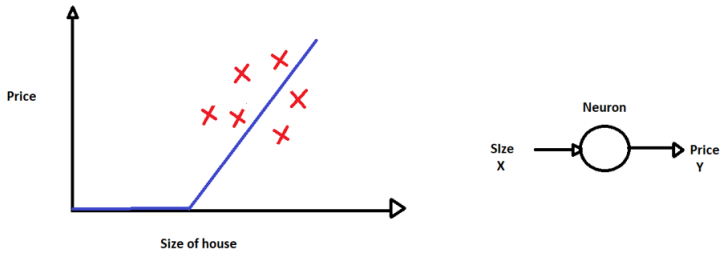

Figure 2.1: Simple predicting function using regression

For an example, for prediction of housing price, we have a data set of six houses where the size of the house in per square feet and prices of the house is given. If we enroll the data in a graph, we will get the blue regression line which is the function for predicting price of the house. The price can not be negative and so the curve is bent where it ends up zero. It is a very simple neural network. Here, the input of the neural network is the size of a house, x. It goes into the little circle node which is a neuron. Then it computes this linear function, it outputs the price, y. This is a single neuron example. A larger neural network is formed by inputting many single neurons and stacking them together. In this example, if we add more attribute like no of rooms, area, this will mean adding of more single neurons. These attributes will be considered as input.

2.1 Supervised and Unsupervised Learning

Neural network is an approach within the field of machine learning, which means, constructing algorithms that solves problems and make predictions. The learning methodology of the algorithm is classified into supervised and unsupervised learning.

Supervised learning is a method that is already provided with both input and output. The input data targets the specific output data and the algorithm learns the mapping procedure. For an example, if pictures of car is given as input. After training the model we have to determine whether it is a picture of car or not from a new picture. Supervised learning is mostly used for classification and regression.

Whereas in unsupervised learning there is no corresponding target output of the input and the model is expected to determine them by itself. To illustrate, it is a sequence of input is provided but no corresponding output values. For an example, from pictures of vehicles, machine automatically categorize them analyzing their size, shape, color etc. Unsupervised learning is used for clustering and association.

2.2 Data structure

In neural networks, data can be both structured data and unstructured data. Structured data is basically databases of data where all the features or attributes are in proper order in different column and they have defined meaning. In contrast, unstructured data are things like audio or images where the features can be the pixel values of an image or the individual words in a piece of text. It is harder to deal with for an unstructured data for a machine compared to structured data because of its unorganized structure. Whereas humans are excellent in understanding and interpreting unstructured data. With the growing improvement of neural networks, machines are also getting better at interpreting unstructured data.

2.3 Scale drives deep learning progress

In recent days neural network has been working very well then other methods. To understand the reason, a plotted figure is in below.
In the figure, the horizontal axis represents the amount of and the vertical axis represents the performance of learning algorithms. The performance of a traditional learning algorithm such as SVM or logistic regression as a function of the amount of data will give a curve where improvement of performance will be shown. But if we increase amount of data it will tend to give horizontal line. With the blessing of technology, the number of data is increasing exponentially. Traditional learning algorithm were not able to effectively take advantage of the huge amount of data. If we train a small neural net then performance will be better than traditional learning algorithm. The more we increase the size of neural network the better result we get. For high level performance we need to be able to train a big enough neural network in order to take advantage of the huge amount of data and second, we need to have a lot of data.
Here, scale represents both size of the neural network and size of data. Big size

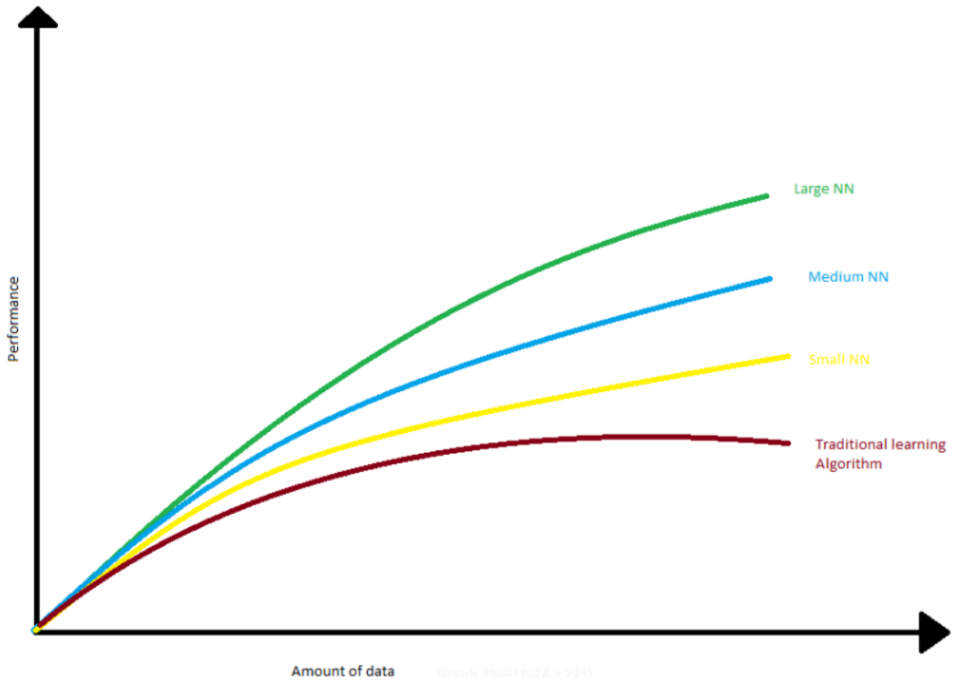

Figure 2.2: Performance of different machine learning algorithm with increasing amount of data

of the neural network means a new network with a lot of hidden units, parameters and connections. The size of the training set depends on size of the given data. For smaller training sets the relative ordering of the algorithms is not very well defined. So, for smaller training data, hand engineering features are needed for better result. And for large training set we see largely neural network dominating other approaches. This is why neural networks is taking off so well.

2.4 Logistic Regression

Logistic regression is a learning algorithm where we determine output label Y (whether 0 or 1) for an supervised learning binary classification problem. For the previous example, for an input image, determining whether it is a car or not.
For given image, feature vector is x.
The predicted output is denoted by y‘which is need to be the probability of y is equal to 1 given the input features x.

$$y‘ = P(y = 1, x), 0 < y‘ < 1 \tag{2.1}$$

So, y‘is the chance of the picture being picture of a car.

Given, x$\in R^n{}_x$,
Parameters of logistic regression: w$\in R^n{}_x$, b$\in R$
For linear regression,

$$y‘ = (w^T + b) \tag{2.2}$$

For binary classification, it is not a good algorithm. As y‘is the chance that y is

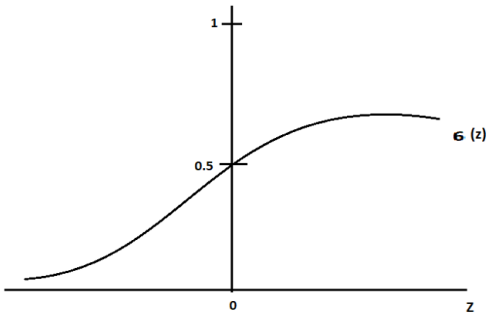

Figure 2.3: Sigmoid function

equal to 1 so the range should be $0 \leq y` \leq 1$. But, it is difficult to enforce that as $w^T x + b$ can be greater than 1 or negative which contradicts with the probability. So in logistic regression, we use sigmoid function[15] -

$$y` = (w^T + b) \tag{2.3}$$

$$or, y` = \sigma(z) \tag{2.4}$$

Here, the formula for the sigmoid function is-

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.5}$$

For large value of z, e^{-z} is close to zero. So,

$$\sigma(z) \approx \frac{1}{1 + 0} = 1 \tag{2.6}$$

If z is a large negative number, e^{-z} is a very large number.So,

$$\sigma(z) \approx \frac{1}{1 + (Large\ number)} \approx 0 \tag{2.7}$$

While implementing logistic regression, we need the value of parameter w and b which we need to learn to calculate the chance of y being equal to one.

Cost function measures the performance of a single training example. To elaborate, it measures the performance of the value of parameter w and b on the training set. So we need to define a cost function in order to train the parameters W and B in the logistic regression model.

From logistic regression,

$$y`(i) = \sigma(w^T x^{(i)} + b); where(z^{(i)}) = \frac{1}{1 + e^{-z(i)}} \tag{2.8}$$

So to learn parameters, a given training set of m training examples is { (x^1, y^1),(x^2,

y^2),. . . ,(x^m, y^m) }. We want the output to be $y^{`i} \approx y^i$.
Here y^i is the ground truth level from the training set.

$$Lossfunction, L = (y`, y) = \frac{1}{2}(y` - y)^2 \tag{2.9}$$

But it does not work well in logistic regression. As there appears an optimization problem that is non convex while learning the parameters. The problem is multiple local optima. As a result, gradient decent might not find global optimum. So, we need to use a different loss function which will give us an optimization problem that is convex. [15]
If we want the algorithm to interpret output, y`as the p(y = 1 | x).

$$Ify = 1; p(yjx) = (y`) \tag{2.10}$$

$$Ify = 0; p(yjx) = (1 - y`) \tag{2.11}$$

$$So, p(yjx) = y`^y(1 - y`)^{(1-y)} \tag{2.12}$$

$$Or, logp(yjx) = log(y`^y(1 - y`)^{(1-y)}) \tag{2.13}$$

$$Or, logp(yjx) = (ylogy` + (1 - y)log(1 - y`)) \tag{2.14}$$

$$Or, L(y`, y) = (ylogy` + (1 - y)log(1 - y`)) \tag{2.15}$$

$$Lossfunction, L(y`, y) = -(ylogy` + (1 - y)log(1 - y`)) \tag{2.16}$$

In logistic regression, we want to make the probability large. So, maximization of the log of the probability corresponds minimization of loss function.
For the first loss function, the squared error has to be as small as possible. To elaborate this-

If y=1 then

$$L(y`, y) = -(ylogy` + (1 - 1)log(1 - y`)) \tag{2.17}$$

$$= -(ylogy` + 0) \tag{2.18}$$

$$= -ylogy`. \tag{2.19}$$

In learning process, for small loss function, - y log y` need to be as big as possible and so y` needs to be large. But in sigmoid function, it can not be bigger than 1.

If y=0 then

$$L(y`, y) = -(0 \times logy` + (1 - 0)log(1 - y`)) \tag{2.20}$$

$$= -log(1 - y`) \tag{2.21}$$

So, - log(1-y`) need to be as big as possible and so y` needs to be small.

For m training example,
P(all labels in training set) =

$$\prod_{i=1}^{m} p(y^{(i)}jx^{(i)}) \tag{2.22}$$

Or, log P(all labels in training set) =

$$\frac{1}{m}\sum_{i=1}^{m} p(y^{(i)}jx^{(i)}) \tag{2.23}$$

$$= -\frac{1}{m}\sum_{i=1}^{m} L(y^{\prime i}, y^i) \tag{2.24}$$

According to maximum likelihood estimation, it should choose the parameters which will maximize the probability. This will lead to cost function. Cost function measures the performance of entire training set. As the cost function has to be minimize to maximize the probability, we eliminate the negative sign.[16]

$$Costfunction, J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(y^{\prime i}, y^i) \tag{2.25}$$

$$= -\frac{1}{m}\sum_{i=1}^{m} [y^i logy^{\prime i} + (1-y^i)log(1-y^{\prime i})] \tag{2.26}$$

Cost function defines the costs of parameters. While training in logistic regression, the value of w and b should be determined where cost function gives minimum value which leads to maximum probability. So, this logistic regression has been turned out as a very small neural network.

2.5 Gradient Decent

Gradient decent algorithm is used to determine the value of parameter w and b on a training set, for which the cost function is as small as possible.

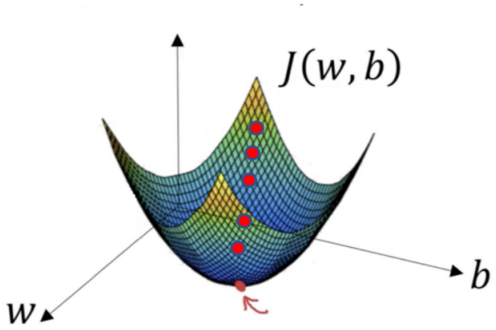

Figure 2.4: Gradient descent

In this diagram, the horizontal axes represent the parameter w and b from training set. The cost function is represented as a surface above parameter w and b. it means that cost function is the height of the surface from a specific point which is a convex function. This being a convex function is one of the main reasons for using this cost function in logistic regression.

At first, w and b are initialized to some initial value. There are many methods for

initializing. For logistic regression. we can use any of them. In convex function, it does not matter where the initial point is as long as it can reach to the same point or close to same point. After the initialization, the first iteration takes place. It takes a step to the steepest downhill direction. The same iteration of taking a step to the steepest downhill direction continues till it finds the global optimum or close to it.

For an example, from the plot, to minimize the function J(w) (ignoring b), the gradient decent works like this. It updates the value of w in each iteration until the algorithm is converged.

$$w := w - \alpha \frac{dJ(w)}{dw} \tag{2.27}$$

Where α = learning rate which control how big the step will be in each iteration

dJ(w)$\frac{}{dw}$ = change of w, slope of the function

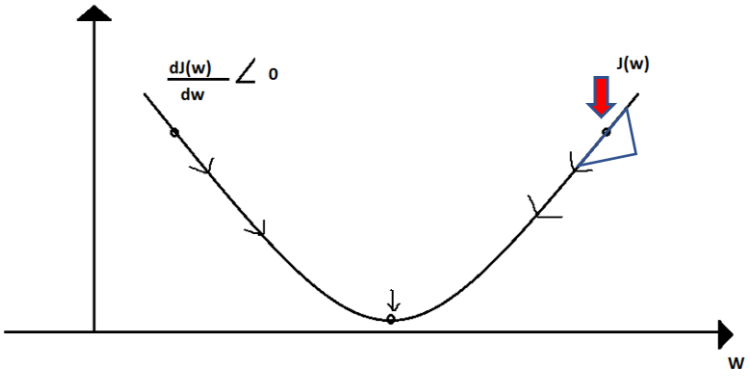

Figure 2.5: Process of finding minimum in gradient decent

As derivative means slope of a junction so at the marked point (red arrow) J(w) is the triangle at the tangent at that point. So, the value of derivative is positive.

$$w := w - \alpha \frac{dJ(w)}{dw} \tag{2.28}$$

As long as the derivative is positive, it will keep subtracting from w which will take the curve to a down-left step.

If the point is in blue arrow, then the value of derivative is negative.
So, it will be

$$w := w + \alpha \frac{dJ(w)}{dw} \tag{2.29}$$

So, it will keep adding to w and the value of w will increase.
Wherever the point is taken, it will gradually move to the global minimum.

Now, for gradient decent to work properly so that it can take these steps of steepest

decent, we need to know the value of $\frac{dJ(w)}{dw}$, slope of the function. Previously, we wrote the function only for w ignoring b for example purpose. but we need to have both w and b for logistic regression.
For J (w, b),

$$w := w + \alpha \frac{dJ(w,b)}{dw} \tag{2.30}$$

$$b := b + \alpha \frac{dJ(w,b)}{db} \tag{2.31}$$

This is how gradient decent is represented.

From logistic regression, we can say that-

$$z = w^T x + b \tag{2.32}$$

$$y' = a = \sigma(z) \tag{2.33}$$

$$L(a, y) = -(yloga + (1 - y)log(1 - a)) \tag{2.34}$$

If we put this in computation graph for two features x_1 and x_2, it will look this.

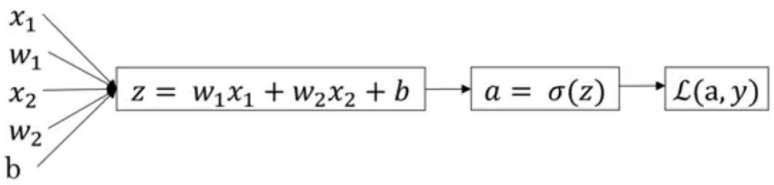

Figure 2.6: Steps of logistic regression

For corresponding x_1 and x_2, w_1, and w_2 is taken. To reduce the loss function, we need to modify w and b. To determine the loss function, we need to compute the backward propagation.
To compute the derivative of loss function, L (a, y), we have to go a step back. Then a derivation of the loss function in respect to da has to be performed.[17]

After the computation of da with respect of a, we need to go a step back to compute dz. Here we'll compute loss function in respect with dz. And the final step of the computation is to compute how much we need to change w and b.

$$\frac{d\,\mathrm{L}}{d\mathrm{w1}} = dw_1 \ = x_1 \ dz \tag{2.35}$$

$$dw_2 \ = x_2 \ dz \tag{2.36}$$

$$db = dz \tag{2.37}$$

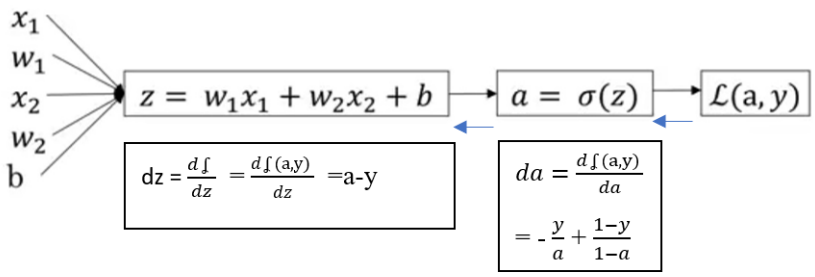

Figure 2.7: Steps of computing loss in logistic regression

$$So, w_1 := w_1 - \alpha dw_1 \tag{2.38}$$

$$w_2 := w_2 - \alpha dw_2 \tag{2.39}$$

$$b := b + \alpha db \tag{2.40}$$

This is gradient decent for a single training example for logistic regression. Now we will look at gradient decent for m training example.

$$Costfunction, J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(z^{(i)}, y^{(i)}) \tag{2.41}$$

$$a^{(i)} = y^{`(i)} = \sigma(w^T x^{(i)} + b) \tag{2.42}$$

$$So, \frac{d}{dw1} J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{d}{dw1} L(a^{(i)}, y^{(i)}) \tag{2.43}$$

To implement the gradient descent, the algorithm below from the previous calculation will help to determine overall gradient while using logistic regression.

[h]
J=0; $\mathrm{dw}_1 = 0; dw_2 = 0; db = 0$
$i = 1 \to m$
$z^{(i)} = w^T x^{(i)} + b$
$a^{(i)} = \sigma(z^{(i)})$
$J+ = -[y^{(i)} log a^{(i)} + (1 - y^{(i)}) log(1 - a^{(i)})]$
$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 += x^{(i)}{}_1 dz^{(i)}$
$dw_2 += x^{(i)}{}_2 dz^{(i)}$
$db += dz^{(i)}$

$J/ = m$
$dw_1/ = m$
$dw_2/ = m$
$db/ = m$
Here, we have used dw_1 and dw_2 accumulators to sum over the entire training set and dz as one training example. By determining the value of dw_1 ,dw_2 and db, we can determine w_1, w_2 and b.

$$w_1 := w_1 - \alpha dw_1 \tag{2.44}$$

$$w_2 := w_2 - \alpha dw_2 \tag{2.45}$$

$$b := b + \alpha db \tag{2.46}$$

But there is a limitation if we implement logistic regression like this. We have to take two for loops for this, one is for m training example and second one is to determine dw_1 ,dw_2,... dw_n (for n features). According to rule of thumb, these explicit for loops make the algorithm less efficient. It will create problem and take a lot of time, if the scale of data is higher. We can solve this problem of explicit for loop by introducing vectorization.

2.6 Vectorization

If we use non vectorized version, for larger amount if data it will take a lot of time. So, the efficiency will be reduced. Using the vectorized version, we can perform the same thing much faster.

In logistic regression,

$$z = w^T x + b \tag{2.47}$$

For non-vectorized implementation:
z=0
i in range (n,x)
z+= w[i] * x[i]
z+=b
For vectorized implementation:

$$Z = np.dot(w, x) + b \tag{2.48}$$

If we look at the code, we can see both vectorized and non-vectorized version has been implemented there. We can see from the output that the non-vectorized version takes a lot of time than vectorized one. Vectorized one is 46 times faster than the non-vectorized one.
So, if we use vectorization in deep learning algorithms, it will work very fast.
In logistic regression gradient descent implementation, it needs two for loop. To eliminate explicit for loop using vectorization, instead of initializing dw_1, dw_2 ,...., dw_n to zero we will use dw=np.zeroes(n-x,1)

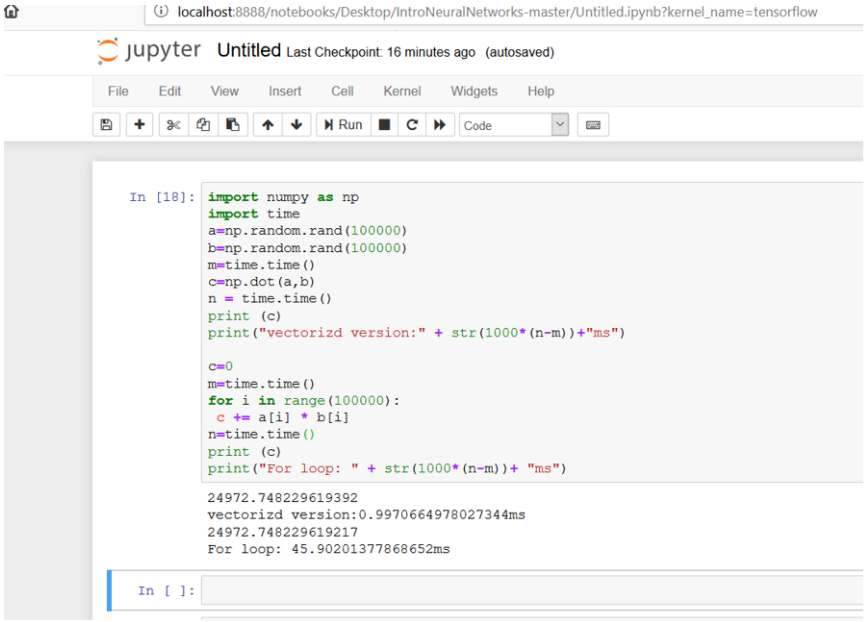

Figure 2.8: Computational time of vectorized implementation

The updated algorithm will be:
J=0; dw=np.zeroes(n-x,1) ; db=0
i=1 $\rightarrow m$
z$^{(i)} = w^T x^{(i)} + b$
a$^{(i)} = \sigma(z^{(i)})$
J+= - [y$^{(i)}$log$a^{(i)}$ + $(1 - y^{(i)})log(1 - a^{(i)})$]
dz$^{(i)} = a^{(i)} - y^{(i)}$
dw+= x$^{(i)}dz^{(i)}$
db += dz$^{(i)}$
$J/ = m$
dw /=m
db /=m This is how vectorization eliminates one explicit loop of the algorithm, which makes the algorithm run much faster.
From previous discussion, we know how to use vectorization to compute prediction. To use vectorization in gradient decent, suppose we take m training example.
For the gradient computation, we know,

For the first training example,

$$dz^{(1)} = a^{(1)} - y^{(1)} \tag{2.49}$$

For the second training example,

$$dz^{(2)} = a^{(2)} - y^{(2)} \tag{2.50}$$

.
.
.

For the m-th example,

$$dz^{(m)} = a^{(m)} - y^{(m)} \tag{2.51}$$

$$A = [a^{(1)} \quad a^{(2)} \quad \ldots\ldots a^{(m)}]; Y = [y^{(1)} \quad y^{(2)} \quad \ldots\ldots y^{(m)}] \tag{2.52}$$

$$dZ = [dz^{(1)} \quad dz^{(2)} \quad \ldots\ldots dz^{(m)}] \tag{2.53}$$

$$= [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \quad \ldots\ldots a^{(m)} - y^{(m)}] \tag{2.54}$$

In the previous implementation, we saw-
For w:

$$dw = 0 \tag{2.55}$$

$$dw+ = x^{(1)}dz^{(1)} \tag{2.56}$$

$$dw+ = x^{(2)}dz^{(2)} \tag{2.57}$$

$$\vdots$$

$$dw+ = x^{m}dz^{(m)} \tag{2.58}$$

$$dw/ = m \tag{2.59}$$

For b:

$$db = 0 \tag{2.60}$$

$$db+ = db^{(1)} \tag{2.61}$$

$$db+ = db^{(2)} \tag{2.62}$$

$$\vdots$$

$$db+ = dz^{(m)} \tag{2.63}$$

$$db/ = m \tag{2.64}$$

we can eliminate this by-

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} \tag{2.65}$$

$$dw = \frac{1}{m}Xdz \tag{2.66}$$

$$= \frac{1}{m}[^{(1)} \quad x^{(2)} \quad \ldots\ldots x^{(m)} \,] \begin{bmatrix} \mathrm{dz(1)} \\ \vdots \\ \mathrm{dz(m)} \end{bmatrix} \tag{2.67}$$

$$= \frac{1}{m}[x^{(1)}dz^{(1)} \quad x^{(2)} \; dz^{(2)} \; \ldots\ldots x^{(m)} \; dz^{(m)}] \tag{2.68}$$

This is how we can determine parameter dw and db value. Now, we can implement this in logistic regression algorithm. Using this, we can eliminate the remaining 1 for loop and make the algorithm more efficient.
Z = $\mathrm{w}^T X + b$
= np.dot(w.T,X) +b
A = $\sigma(z)$

dZ = A- Y
dw = $\frac{1}{m}$ X dz
db= $\frac{1}{m}$ np.sum(dZ)
w := w - dw
b := b - db

Thus, we can perform both forward and back propagation using vectorization in logistic regression gradient decent using m training examples most efficiently.

2.7 One Hidden Layer Neural Network

In neural network, the input values, x^1, x^2, x^3 .... are stacked up vertically. This is known as input layer. After input layer there is another layer that work to train sets which is known as hidden layer. And the last layer is the output layer.
In logistic regression, we computed z which was used to compute a, activation that contains a sigmoid function. It Is implemented on as single node. But in neural network multiple node is used where in each node a sigmoid function of logistic regression is defined. So, neural network is formed by stacking together a lot of sigmoid units.

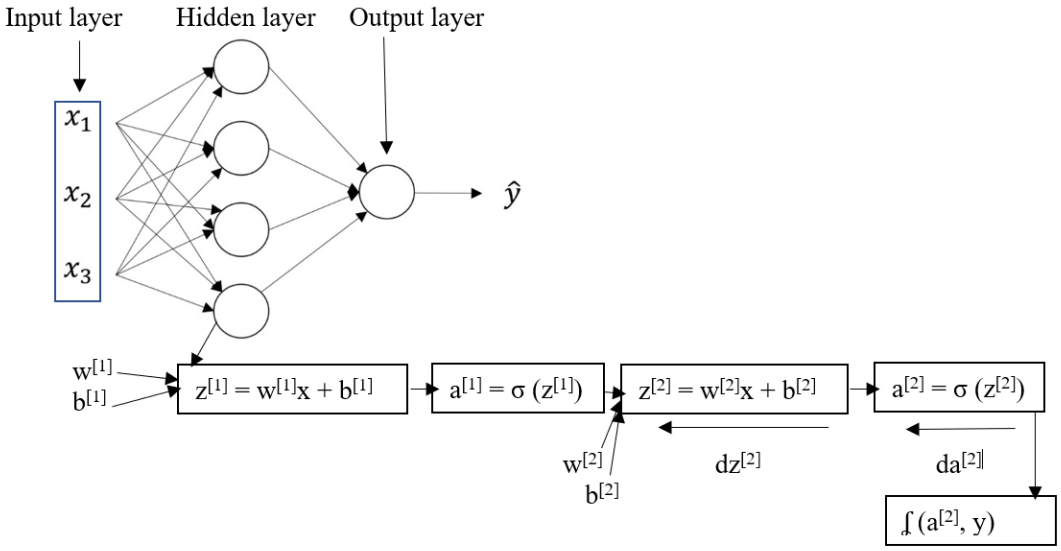

Figure 2.9: Computation in different node of neural network

Like logistic regression, the nodes will compute z and then using that compute a which stands for activation. This computation will be held sequentially one after another according to representation of layer. Here, "[1]" means layer 1, "[2]" means layer 2 and so on. So, first it will compute for layer 1 then layer 2 and so on. The value of first layer's x will be scratched from given input vector. But for further layers, value of x will be derived from previous layer.
In the hidden layer, activation, a is a 4×1 matrix for this example as for 4 node, 4 activation $a^{[1]}{}_1$, $a^{[1]}{}_2$, $a^{[1]}{}_3$ .....will be produced. The w vector is will be m×n matrix where m is number of inputs and n is number of layers. But in output layer

w vector is will be 1×m matrix.
Then finally loss function calculation occurs at end. This is for forward propagation. For back propagation we have to determine da, dz sequentially to get the value of dw and db which will make the cost function smallest.

2.8 Computation of Neural Network

Each node in neural network perform two step computation like logistic regression. For an example, we consider only one node of the hidden layer of the figure.

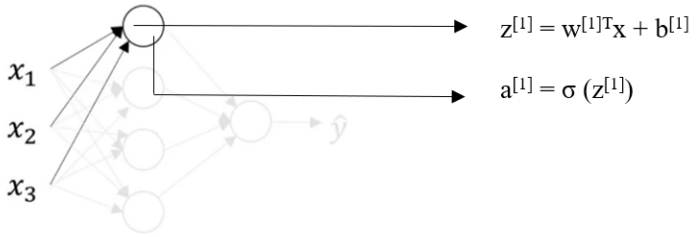

Figure 2.10: Computation in layer 1 node 1

Here, first it computes $z^{[1]}$ and using that value determines $a^{[1]}$. This process repeats in all of the nodes of hidden layer.

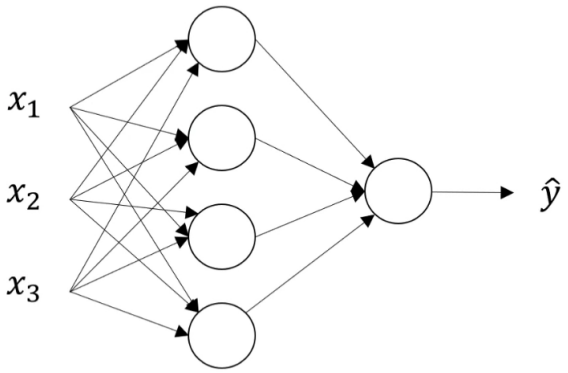

Figure 2.11: Neural network with 1 hidden layer

Node 1:

$$z_1{}^{[1]} = w_1{}^{[1]T}x + b_1{}^{[1]} \; ; a_1{}^{[1]} = \sigma(z_1{}^{[1]}) \tag{2.69}$$

Node 2:

$$z_2{}^{[1]} = w_2{}^{[1]T}x + b_2{}^{[1]} \; ; a_2{}^{[1]} = \sigma(z_2{}^{[1]}) \tag{2.70}$$

Node 3:

$$z_3{}^{[1]} = w_3{}^{[1]T}x + b_3{}^{[1]} \; ; a_3{}^{[1]} = \sigma(z_3{}^{[1]}) \tag{2.71}$$

Node 4:

$$z_4{}^{[1]} = w_4{}^{[1]T}x + b_4{}^{[1]} \; ; a_4{}^{[1]} = \sigma(z_4{}^{[1]}) \tag{2.72}$$

To calculate this, using for loops will not be efficient. So, we will apply vectorization here.

$$\mathrm{Z}^{[1]} = \begin{bmatrix} \mathrm{z1[1]} \\ \mathrm{z2[1]} \\ \mathrm{z3[1]} \\ \mathrm{z4[1]} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{w1[1]T} \\ \mathrm{w2[1]T} \\ \mathrm{w3[1]T} \\ \mathrm{w4[1]T} \end{bmatrix} \begin{bmatrix} \mathrm{x1} \\ \mathrm{x2} \\ \mathrm{x3} \\ \end{bmatrix} + \begin{bmatrix} \mathrm{b1[1]} \\ \mathrm{b2[1]} \\ \mathrm{b3[1]} \\ \mathrm{b4[1]} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{w1[1]Tx} \; + \; \mathrm{b1[1]} \\ \mathrm{w2[1]Tx} \; + \; \mathrm{b2[1]} \\ \mathrm{w3[1]Tx} \; + \; \mathrm{b3[1]} \\ \mathrm{w4[1]Tx} \; + \; \mathrm{b4[1]} \end{bmatrix}$$

$$\mathrm{a}^{[1]} = \begin{bmatrix} \mathrm{a1[1]} \\ \mathrm{a2[1]} \\ \mathrm{a3[1]} \\ \mathrm{a4[1]} \end{bmatrix} = \; (\mathrm{Z}^{[1]})$$

This is how vectorization works on neural network. Now, the $\mathrm{a}^{[1]}$ of the hidden layer will go to output layer as an input vector and compute the same thing. So, the final algorithm will only consider this:

$$z^{[1]} = W^{[1]} \;\; x + b^{[1]} \tag{2.73}$$

(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z_1{}^{[1]}) \tag{2.74}$$

(4,1) (4,1)

$$z^{[2]} = W^{[2]} \; a^{[1]} + b^{[2]} \tag{2.75}$$

(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]}) \tag{2.76}$$

(1,1) (1,1)

This is how simply we can create a 1 hidden layer neural network for a single training set using just these 4 equations. For multiple training set, it will be slightly different. For input vector x, output is

$$a^{[2]} = y`; \tag{2.77}$$

For 1^{st} training example, for input vector $x^{(1)}$, output is

$$a^{[2](1)} = y`^{(1)} \tag{2.78}$$

For 2^{nd} training example, for input vector $x^{(2)}$, output is

$$a^{[2](1)} = y`^{(2)} \tag{2.79}$$

.
.
.

For m-th training example, for input vector $x^{(m)}$, output is

$$a^{[2](m)} = yM \tag{2.80}$$

For non-vectorized version the algorithm will be:
i=1 $\rightarrow m$,
$z^{[1](i)} = W^{[1]T}\ x^{(i)} + b^{[1]}$
$a^{[1]\ (i)} = (z_1{}^{[1]\ (i)})$
$z^{[2]\ (i)} = W^{[2]T}\ a^{[1]\ (i)} + b^{[2]}$
$a^{[2]\ (i)} = (z^{[2]\ (i)})$
To eliminate the for loop and make it vectorize:

$$X = [x^{(1)}\ \ x^{(2)}\ \ .....x^{(m)}] \tag{2.81}$$

This is a $n_x \times m$ matrix.

$$Z^{[1]} = [z^{[1](1)}\ \ z^{[1](2)}\ \ .....z^{[1](\ (m)}] = W^{[1]T}X + b^{[1]} \tag{2.82}$$

$$A^{[1]} = [a^{[1](1)}\ \ a^{[1](2)}\ \ .....a^{[1](m)}] = \sigma\ (Z^{[1]}) \tag{2.83}$$

$$Z^{[2]} = [z^{[2](1)}\ \ z^{[2](2)}\ \ .....z^{[2](\ (m)}] = W^{[2]T}A^{[1]} + b^{[2]} \tag{2.84}$$

$$A^{[2]} = [a^{[2](1)}\ \ a^{[2](2)}\ \ .....a^{[2](m)}] = \sigma\ (Z^{[2]}) \tag{2.85}$$

This A is an $m \times n$ matrix where m is number of training examples and n is number of nodes in hidden layer. Same intuition works for both matrix Z and X. Thus, we create a single hidden layer neural network for m training examples using vectorization to make the process faster.

2.9 Activation Function

2.9.1 Activation Function

While building a neural network, one of the important things is decide which activation function should be used in hidden layer and output layer. Previously, we have been giving examples of neural network using sigmoid function as activation function. But sometimes it's not efficient enough to use sigmoid function. Instead of sigmoid function '', we can use different function such as a nonlinear function 'g'. We can also use different activation function in different layer.
The limit of sigmoid function being 0 to 1 makes it less efficient. On the other hand, hyperbolic tangent function, tanh mostly works better than sigmoid function. Its limit being -1 to +1 has make it work better than sigmoid function.

$$a = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.86}$$

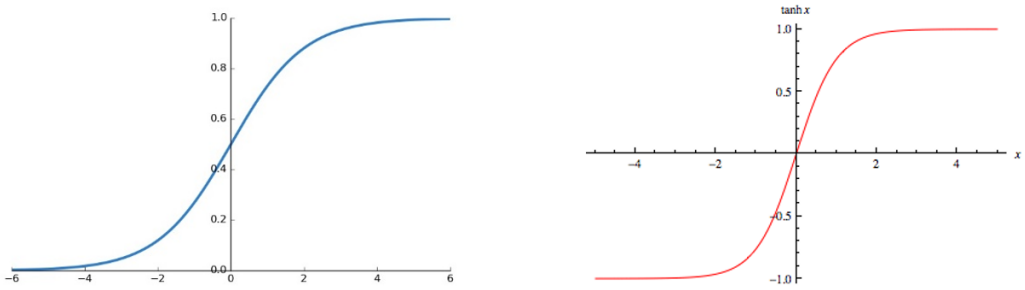

Figure 2.12: Sigmoid and Tanh activation function

The mean of tanh function is closer to 0. For centered data, if it always gives mean 0 then it helps the next layer to work faster. In every way tanh function works better than sigmoid function except for in output layer. For output layer, y can be either 0 or 1. So, y' need to be in range of $0 \leq y' \leq 1$ for determining output rather than -1 to 1. This is why sigmoid function should always be used in output layer. A disadvantage of sigmoid and tanh function is that if z is too big or too small, slope of the function turns out to be close to 0 that makes the gradient decent slower.

Another efficient activation function is the ReLU function.

$$a = max(0, z) \tag{2.87}$$

If z is positive then the derivative is 1 and if z is negative then derivative is 0. For binary classification, ReLU function is the most used activation function. There is another ReLU function known as leaky ReLU.

$$a = max(0.001z, z) \tag{2.88}$$

As the value of z being negative makes the derivative 0 in ReLU function, leaky ReLU takes a slight slope. The use of ReLU and leaky ReLU makes neural network algorithm much faster as the slope of the function is different from 0.[18]

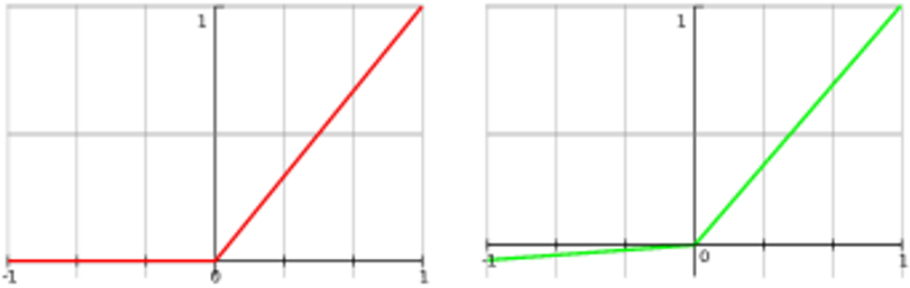

Figure 2.13: ReLU and leaky ReLU activation

These are the activation function used in neural network algorithm. The performance of the activation function may vary from code to code. So, while implementing algorithm, it is best to try them all to compare results.

For neural network algorithm we always use non linear activation function rather than linear function. If we used linear function, then the neural network would just produce an output that is linear function of the input. Even if we use a lot of hidden layer, it will just compute a linear function. Apart from some very special cases, use of linear activation function in neural network is rare. Mostly non-linear activation function is used for its better performance.

2.9.2 Derivative of Activation function

For implementing back propagation, we have to compute the slope of the derivative of activation function. The computation of derivative of activation functions is given below.

Sigmoid Function:

$$g(z) = \frac{1}{1+e^{-z}} \frac{d}{dz} g(z) \tag{2.89}$$

This is the slope of g(z) at z.

$$= \frac{1}{1+e^{-z}} (1 - \frac{1}{1+e^{-z}}) \tag{2.90}$$

$$= g(z)(1 - g(z)) \tag{2.91}$$

If z=10 (large number),

$$g(z) \approx 1; \tag{2.92}$$

$$\frac{d}{dz} g(z) \approx 1(1-1) \approx 0 \tag{2.93}$$

If z= -10 (small number),

$$g(z) \approx 0; \tag{2.94}$$

$$\frac{d}{dz}g(z) \approx 0(1-0) \approx 0 \tag{2.95}$$

If z=0,

$$g(z) \approx \frac{1}{2}; \tag{2.96}$$

$$\frac{d}{dz}g(z) \approx \frac{1}{2}(1-\frac{1}{2}) \approx \frac{1}{4} \tag{2.97}$$

This is the correct value of slope of the derivative of the function in z=0.

Tanh Function:

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.98}$$

$$\frac{d}{dz}g(z) \tag{2.99}$$

$$= 1 - (tanh(z))^2 \tag{2.100}$$

If z=10 (large number),

$$tanh(z) \approx 1; \tag{2.101}$$

$$\frac{d}{dz}g(z) \approx 0 \tag{2.102}$$

If z= -10 (small number),

$$tanh(z) \approx -1; \tag{2.103}$$

$$\frac{d}{dz}g(z) \approx 0 \tag{2.104}$$

If z=0,

$$tanh(z) \approx 0 \tag{2.105}$$

$$\frac{d}{dz}g(z) \approx 1 \tag{2.106}$$

ReLU Function:

$$g(z) = max(0, z) \tag{2.107}$$

If z<0,

$$g(z) = 0 \tag{2.108}$$

If z>0,

$$g(z) = 1 \tag{2.109}$$

If z=0, g(z) is undefined

Leaky ReLU Function:

$$g(z) = max(0.01z, z) \tag{2.110}$$

If z<0,

$$g(z) = 0.01 \tag{2.111}$$

If z>0,

$$g(z) = 1 \tag{2.112}$$

These are the computation of derivative of four activation function which is needed in back propagation.

2.10 Gradient decent for Neural Network

We can implement gradient decent for creating 1 hidden layer neural network. For that lets assume, number of nodes in input layer is $n^{[0]}$, number of nodes in input layer is $n^{[1]}$ and number of nodes in input layer is $n^{[0]}$. Now the parameters for the algorithm will be $w^{[1]}$ ($n^{[1]} \times n^{[0]}$ matrix), $b^{[1]}$ ($n^{[1]} \times 1$ matrix), $w^{[2]}$ ($n^{[2]} \times n^{[1]}$ matrix) and $b^{[1]}$ ($n^{[2]} \times 1$ matrix).
Both forward and back propagation calculation is given below to make a 1 hidden layer neural network with gradient decent.

Forward Propagation:

First step:

$$Z^{[1]} = W^{[1]}\ X + b^{[1]} \tag{2.113}$$

Second step:

$$A^{[1]} = g(Z^{[1]}) \tag{2.114}$$

Third step:

$$Z^{[2]} = W^{[2]}\ A^{[1]} + b^{[2]} \tag{2.115}$$

Forth step:

$$A^{[2]} = g(Z^{[2]}) = \sigma(Z^{[2]}) \tag{2.116}$$

Back Propagation:

First step:

$$da = \frac{d}{da}L(a, y) \tag{2.117}$$

$$= -yloga - -(1-y)log(1-a) \tag{2.118}$$

$$= -\frac{y}{a} + \frac{1-y}{1-a} \tag{2.119}$$

Second step:

$$dz\ = \frac{d}{dz}L(a, y) \tag{2.120}$$

$$= \frac{dL}{da}\frac{da}{dz} \tag{2.121}$$

$$= da\frac{dg(z)}{dz} \tag{2.122}$$

$$= da.g`(z) \tag{2.123}$$

$$= a - y \tag{2.124}$$

Third step:

$$dw = dz.x \tag{2.125}$$

Forth step:

$$db = dz \tag{2.126}$$

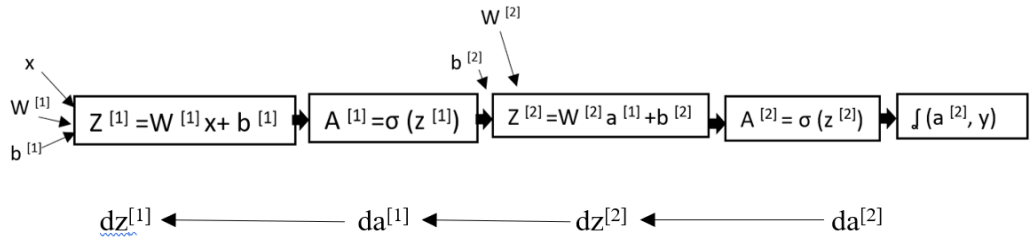

Figure 2.14: Forward and backward computation

For this example, we have to calculate this back propagation steps twice as we will deal with both hidden layer and output layer. So, the process is given below:

For output layer:

$$dz^{[2]} = a^{[2]} - y \tag{2.127}$$

$$dw^{[2]} = dz^{[2]}\ a^{[2]T} \tag{2.128}$$

$$db^{[2]} = dz^{[2]} \tag{2.129}$$

For hidden layer:

$$dz^{[1]} = W^{[2]T}\ dz^{[2]}\ * g^{[1]}(z^{[1]}) \tag{2.130}$$

$$dw^{[1]} = dz^{[1]}\ x^T \tag{2.131}$$

$$db^{[1]} = dz^{[1]} \tag{2.132}$$

This is done for single training example. For m training example, we have to vectorized it.

$$dZ^{[2]} = A^{[2]} - y \tag{2.133}$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]}\ A^{[2]T} \tag{2.134}$$

$$db^{[2]} = \frac{1}{m}\ np.sum(dZ^{[2]}, axis = 1, keepdims = True) \tag{2.135}$$

$$dZ^{[1]} = W^{[2]T}\ dZ^{[2]}\ * g^{[1]}(Z^{[1]}) \tag{2.136}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]}\ X^T \tag{2.137}$$

$$db^{[1]} = \frac{1}{m}\ np.sum(dZ^{[1]}, axis = 1, keepdims = True) \tag{2.138}$$

Here, axis=1 means summing horizontally and keepdims= True means keep python away from outputting a rank 1 array. After finishing forward and back propagation, a one hidden layer neural network for gradient decent taking m input training examples will be created.

2.11 Random Initialization

One of the important things while building a neural network is initializing the parameters. While talking about logistic regression, we initialized the parameters value as 0. But in neural network, if we initialized it as 0, it will not work except for parameter b. Because if we initialize the parameter w as 0 then the activation functions in all the nodes of the hidden layer $a_1^{[1]}$, $a_2^{[1]}$, $a_3^{[1]}$.... will be equal. The reason behind $a_1^{[1]}$=$a_2^{[1]}$= $a_3^{[1]}$=.... is the hidden units are computing exactly same function. And, while doing back propagation, $dz^{[1]}$ and $dz^{[2]}$ will also be equal.

Initializing the parameters zero makes all the hidden units identical as every row carries the same value. So, no matter how many hidden units or how long training process is, it will always compute same function. So, it is better to use 1 hidden unit for this case. We can solve this problem by taking random initialization.

$W^{[1]}$ = np.random.randn((m,n)) * 0.01 ; where it is a m is row and n is column of matrix
$b^{[1]}$ = np.zero((m,n))
$W^{[2]}$ = np.random.randn((m,n)) * 0.01 ; where it is a m×n matrix
$b^{[2]}$ = np.zero((m,n))

Here, b is initialized as zero as it does not create symmetry problem where values of each rows are same.
The reason for multiplying w with 0.01, a small number is that if we take a large number then for $z^{[1]}$= $W^{[1]}$ x +$b^{[1]}$, z will be a large number. While calculating $a^{[1]}$= $g^{[1]}(z^{[1]})$, for bigger value of z, the activation function will be very flat. As the slope of the function will be smaller then gradient will get smaller. This will make the learning slower. This problem is only for sigmoid or tanh function. For other function, it does not create any problem. Although we need this as in output layer, we use sigmoid function. There can be better constant than 0.01 depends on type of the neural network. This value will work better for 1 hidden layer neural network. But for very deep neural network, changing the value will give better performance.

2.12 Deep L-layer Neural network

So far, we have seen neural network with one hidden layer. In neural network, we can use multiple hidden layers. The greater number of hidden layers makes the

neural network deeper. So, a neural network with a lot of hidden layer is called deep neural network.

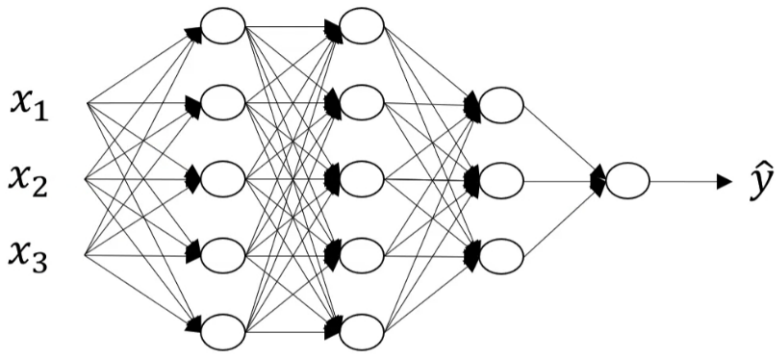

Figure 2.15: Deep neural network

Each hidden layer contains multiple nodes which are called hidden units. For this example, the number of layer, L is 5. And the number of hidden units are denoted as $n^{[l]}$, where l is the layer number. Such as, $n^{[2]}$ =5 defines that second hidden layer has 5 hidden units.
To build a L-layer neural network we have to compute the forward and back propagation.

Forward propagation:

For given input X=$a^{[l-1]}$, we have to determine output $a^{[l]}$ and $z^{[l]}$ by forward propagation.

$$z^{[l]} = W^{[l]}\ a^{[l-1]} + b^{[l]} \tag{2.139}$$

$$a^{[l]} = g^{[l]}\ (z^{[l]}) \tag{2.140}$$

This is the non-vectorized version. To vectorized it:

$$Z^{[l]} = W^{[l]}\ A^{[l-1]} + b^{[l]} \tag{2.141}$$

$$A^{[l]} = g^{[l]}\ (Z^{[l]}) \tag{2.142}$$

Back propagation:

For given input $da^{[l]}$, we have to determine output $da^{[l-1]}$,$dw^{[l]}$ and $db^{[l]}$ by back propagation.

$$dz^{[l]} = da^{[l]}\ * g^{[l]}(z^{[l]}) \tag{2.143}$$

$$W^{[l=1]T} dz^{[l]}\ a^{[l-1]} \tag{2.144}$$

$$db^{[l]} = dz^{[l]} \tag{2.145}$$

$$da^{[l-1]} = W^{[l]T}\ dz^{[l]} \tag{2.146}$$

This is the non- vectorized version. For vectorization:

$$dZ^{[l]} = dA^{[l]}\ * g^{[l]}(Z^{[l]}) \tag{2.147}$$

$$= W^{[l=1]T} = \frac{1}{m} dZ^{[l]}\ A^{[l-1]T} \tag{2.148}$$

$$db^{[1]}\ = \frac{1}{m}\ np.sum(dZ^{[1]}, axis = 1, keepdims = True) \tag{2.149}$$

$$dA^{[l-1]} = W^{[l]T}\ dZ^{[l]} \tag{2.150}$$

This is how a deep neural network with L-layer is implemented. The whole thing is summarized below where in first 2 hidden layer ReLU function has been applied.

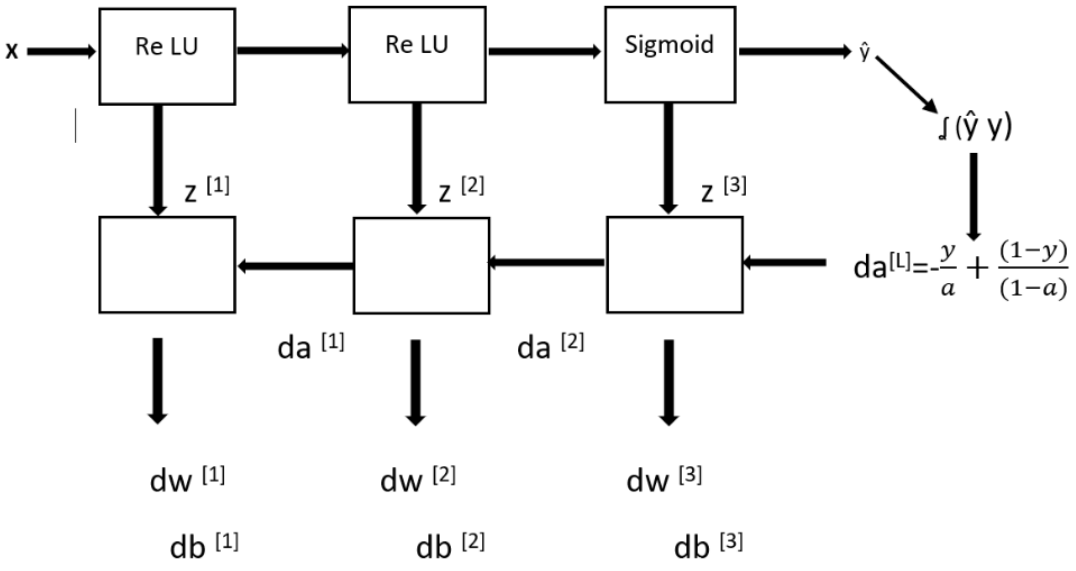

Figure 2.16: Implementation of L-layer neural network

One problem arises in building neural network frequently is the formation of the matrix. If we don't give correct form of matrix, output will be incorrect. So, the

formation of the matrices should be:
$W^{[l]}$: ($n^{[l]}$, $n^{[l-1]}$)
$b^{[l]}$: ($n^{[l]}$, 1)
$Z^{[l]}$, $A^{[l]}$: ($n^{[l]}$, m)
$dZ^{[l]}$, $dA^{[l]}$: ($n^{[l]}$, m)

2.13 Why Deep Neural Network is used?

Recently, it is seen that deep neural network works very well to solve problem. For this, the neural network needs to be very big with a lot of layers. A question may rise what is the reason behind this.
In a deep neural network, the whole works is divided into different layers. The first layers do the simplest works. Then the next layers take the result from previous layers and combine them. And finally, in the last layer a complex function is performed to determine the final output. For an example, in case of the speech recognition algorithms, first layer just learn to detect the low level audio forms. The next layer tries to learn the pitches. Then, the next layer tries to combine them into a word. And finally, the last layer combines the words into sentence. Thus, dividing the work into different layer makes if work efficiently.
If in a neural network, lots of layer is not used even then it can give better outputs by using a lot of hidden units. In deep learning, the less the number of hidden layers, the exponentially more hidden units are need to give better results. A neural network with 5 hidden layers will work as same as a neural network with one layer with lots of hidden units.

Neural network works like a human brain. Seeing a picture, a human mind can easily identify the content of the picture. By training, neural network can also determine the content of a picture.

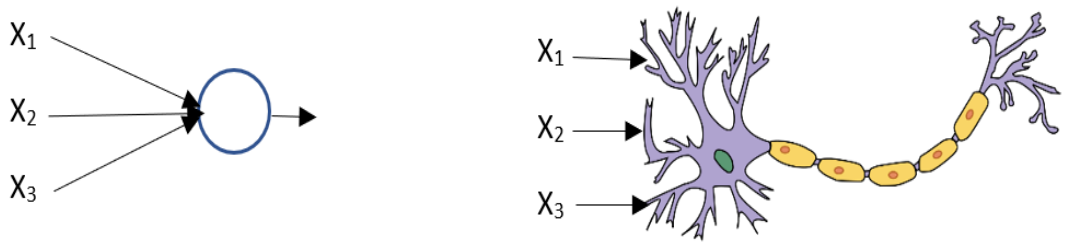

Figure 2.17: Perception vs brain nervous system

In a human brain a simple neuron takes input from outside, process it and then give the output to another neuron. Neural network also works like that. So, as it can work efficiently as close to efficiency of human brain, it has been using more often for solving problem.

2.14 Block Diagram of Building a Neural Network

The whole process of building a neural network is given below. For forward propagation, in every layer, input $A^{[l-1]}$ goes and using $W^{[l]}$ and $b^{[l]}$, it outputs $A^{[l]}$. also a cache $Z^{[l]}$ is produced which is used in the back propagation. And in the back propagation, the derivative of activation function from yl $dA^{[l]}$ is used as an input. Then using $W^{[l]}$,$b^{[l]}$ and $Z^{[l]}$, it outputs $dW^{[l]}$ and $db^{[l]}$.
These values are used to update $W^{[l]}$ and $b^{[l]}$.

$$W^{[l]} := W^{[l]}\ dW^{[l]} \tag{2.151}$$

$$b^{[l]} := b^{[l]}\ db^{[l]} \tag{2.152}$$

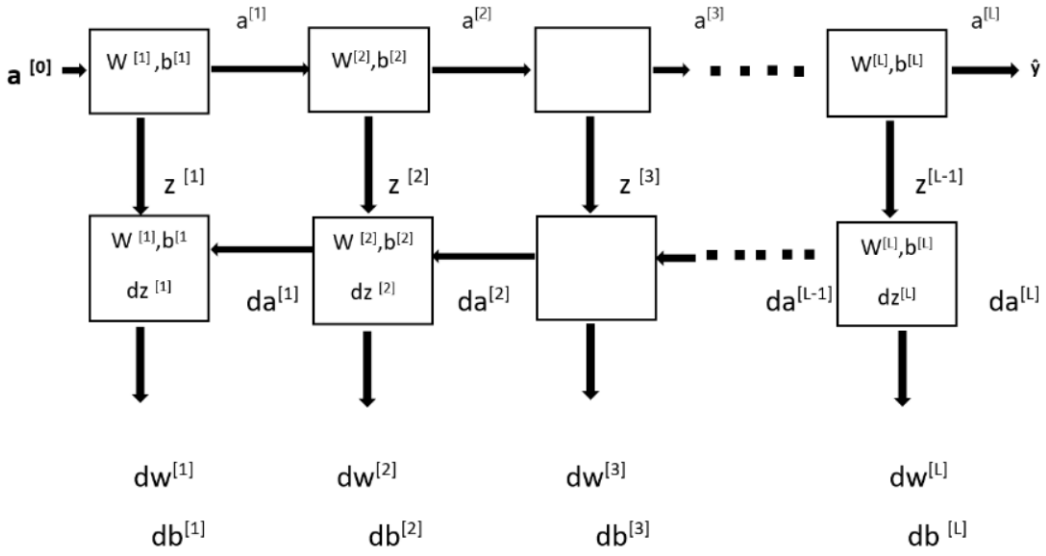

Figure 2.18: Block diagram of neural network

Chapter 3

Improving the Performance of Neural Network

3.1 Hyperparameters

The parameters which control the parameters of neural network W and b, are called hyperparameters. Hyperparameters have to be set in order to determine W and b. Some hyperparameters of neural networks are-

1. Learning rate, α
2. Number of iterations
3. Number of layers
4. Number of hidden units
5. Activation function

The performance of the values of these hyperparameters varies for different structure of neural network. Some value might work well on a neural network but same value might not work well in different neural network. So, while implementing a neural network, it is best to try using different values of hyperparameters and compare the cost function. From that we can easily get which value is working well for that particular neural network. A value of a hyperparameter which works well on a neural network now night not work well after several year on the same neural network as the computation infrastructure keeps changing. So, we have to be careful about selecting the values of these hyperparameter as it can increase or decrease the correctness of the output.

3.2 Data Set

To work with neural network, we have to give some input, x to it. We use data sets as input. Using this data set, a neural network is first trained about the nature of data. In the training session, a neural network learns what should be the output corresponded to an input. Then the neural network is tested to check it how accurately it gives output according to the learning.
So, while taking a data set, we divide it into 3 parts.

1. Train data
2. Development/ hold-out cross validation
3. Test data

In the development set, we determine which model works best for that neural network. From that the best model is used in the test set to check how good the algorithm is performing.
While dividing the data set, we usually take 70% data for training, 20% for development and 10% for test set. But if we take less amount of data then number of development set will be very small in compare to number of training set. We need a good amount of data for development set to run different algorithm model to determine which will work best for it. So, we need to take a bigger data set.

Another issue is distribution of data. The development set test set should be from same distribution. For an example, we want to make a neural network which analyze a picture and determines whether it is a car or not. Suppose, we train a neural network with car pictures taken from webpages and use picture uploaded by programmer in development set. The resolution of picture from webpage and uploaded by programmer will not be same. So, the neural network will not work well. To get the best output, these two set should be come from same distribution.
Usually we don't use the test set. We implement the development set to check the performance of the neural network. This is usually considered as test set.

3.3 Bias and Variance

While training and developing, we determine the error, training set error and development set error. The error level of training set determines the bias. And the error level of development set determines variance. In another word, underfitting set determines bias and overfitting set determines variance. Bias and variance can be two types – high and low.

If we get 1% training set error and 11% development set error, it means the training set has been very well trained but according to that development set hasn't. So, this creates overfitting problem which leads to high variance.

If we get 15% training set error and 16% development set error and if the optimal/bayes error is 0%, it means the training set has not been trained well. So, this creates underfitting problem which leads to high bias.

If we get 15% training set error and 30% development set error and if the optimal/bayes error is 0%, it means the training set has not been trained well. And compare to training set performance, development set's performance is worse. So, this is both high bias and high variance.

If we get 0.5% training set error and 1% development set error, it means both the training set and development set have been very trained well. So, this is both low bias and low variance.

So, by looking at the training set error and development set error, we can determine the bias and variance.

To solve this problem, we first have to check if it is high bias. If we get high bias, to solve it, we need use a bigger neural network with more hidden layers and a lot of hidden units. There are also some neural network algorithms which reduce high bias by itself. So, we can use those too.
After solving high bias, we have to check if it is high variance. If we get high variance, to solve it, we have to import more data or do regularization. And we can also use some neural network algorithm which reduce this problem by itself.[20] Now a days in neural network, there are some tools which eliminate bias variance trade off. This has made the use neural network more popular in supervised learning.

3.4 Regularization

To eliminate the high variance problem, we can use more data. But another efficient method is to use regularization. Previously we have seen the cost function of logistic regression. What regularization does is that it regularizes the parameter of cost function. There are different types of regularization. One of them is L_2 regularization.

L_2 regularization,

$$jjwjj_2{}^2 = \sum_{j=1}^{nx} w_j{}^2 = w^T w \tag{3.1}$$

Cost function,

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) + \frac{\lambda}{2m} jjwjj_2{}^2 \tag{3.2}$$

Here, we can see that we have only regularized parameter w, not b. the reason behind that is w has a greater number of parameters where b has only one parameter.

There is also another regularization known as L_1 regularization.
L_1 regularization cost function,

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) + \frac{\lambda}{2m} jjwjj_1 \tag{3.3}$$

In L_1 regularization, w will have a lot of zeroes so it will be sparse and so it will take less memory in storing the model. But this only helps a bit.
λ is known as regularization parameter which is also a hyperparameter.

This is for logistic regression. For neural network, L_2 regularization will be:

L_2 regularization,

$$jjw^{[l]}jj_F{}^2 = \sum_{i=1}^{n[l-1]} \sum_{j=1}^{n[l]} (w_{ij}{}^{[l]})^2 \tag{3.4}$$

Cost function,

$$J(w^{[1]}, b^{[1]}, \ldots..., w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) + \frac{\lambda}{2m} jjw^{[l]}jj_F{}^2 \tag{3.5}$$

This is known as forbenius norm. Before that, while working on back propagation L_1 regularization will be added with dw.

$$dW^{[l]} = \frac{1}{m}\, dZ^{[l]}\, A^{[l-1]T} + \frac{\lambda}{2m} W^{[l]} \tag{3.6}$$

$$W^{[l-1]} := W^{[l]}\, dW^{[l]} \tag{3.7}$$

$$= W^{[l]} - \frac{\alpha\lambda}{m}\, W^{[l]} - \alpha\frac{1}{m}\, dZ^{[l]} W^{[l-1]} \tag{3.8}$$

Previously, we reduced W by $\alpha\frac{1}{m}$ $\mathrm{dZ}^{[l]}$ $\mathrm{A}^{[\,l-1]T}$. But here, we are reducing W by $\frac{\alpha\lambda}{m}$ $\mathrm{W}^{[l]}$ too. So, it reduces the weight of W. And for that it is called weight decay regularization.

We know that regularization is used to make low variance, to be specify reduce the overfitting problem. If we use the value of very big to reduce overfitting then it will make the values of W matrix closer to zero. This will make most of the hidden units 0 and those hidden units won't give any befit. So, basically it will make the neural network smaller. That will make the network go from overfitting to underfitting which is not quite acceptable. Moreover, making W smaller will make Z smaller as $\mathrm{Z}^{[l]} = \mathrm{W}^{[l]}\ \mathrm{A}^{[l-1]} + \mathrm{b}^{[l]}$. [21]

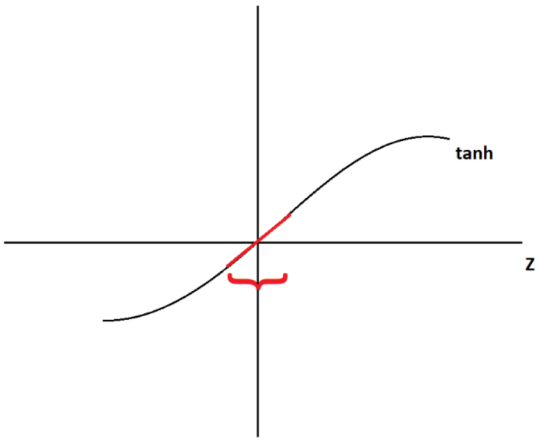

Figure 3.1: Tanh activation function

So, if we take small value of Z then for tanh activation function, it will use only the linear portion. That will make the activation function linear which will lead to making the whole neural network linear. If we could use bigger value of z then the network would be non-linear.

3.5 Dropout Regularization

When dropout regularization is applied in a neural network, it goes through every hidden units and set probability of elimination some hidden units of the neural network. For the example, dropout regularization set some probability and then some of the units are removed their all incoming and outgoing connection. This will make the neural network much smaller. This still does the regularization as this small neural network is still training. To implement dropout regularization, there are several processes. One of them is inverted dropout.

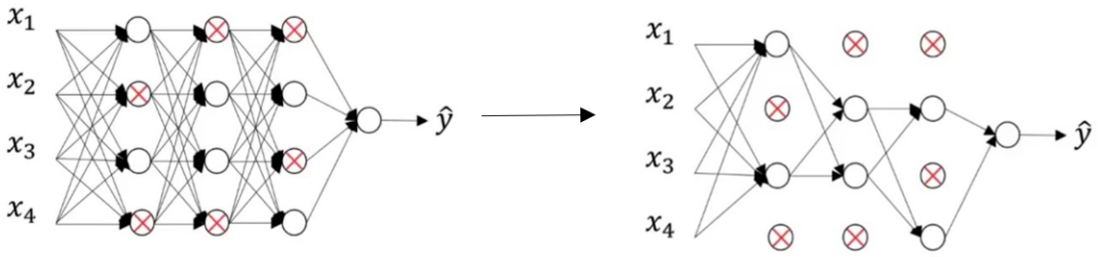

Figure 3.2: Canceling weights using dropout

If inverted dropout is applied in a neural network, lets look a layer L:

$$d^{[l]} = np.random.randn(a^{[l]}.shape[0], a^{[l]}.shape[1]) < keep - prob \tag{3.9}$$

$$a^{[l]} = np.multiply(a^{[l]}, a^{[l]}) \tag{3.10}$$

$$a^{[l]} /= keep - prob \tag{3.11}$$

Here, keep-prob is the probability of keeping a hidden unit and $d^{[l]}$ is the probability of eliminating a hidden unit. $d^{[l]}$ is a random matrix. If $d^{[l]}$ is less than keep-prob the chance of keeping the hidden unit will be 1. If $d^{[l]}$ is greater than keep-prob the chance of keeping the hidden unit will be 0. So, if the activation is multiplied with $d^{[l]}$ being zero, then activation will be 0 which will make the hidden unit eliminated. Dividing $a^{[l]}$ by keep-prob make sure to keep the expected value of $a^{[l]}$ remains same and makes test time easier. So, this inverted dropout eliminates different hidden units going through each layer and make the network smaller.

Another thing is while implementing dropout, randomly any hidden unit can be eliminated. So, the output can't depend on only one feature and for that the weights have to spread out. This is similar to L_2 regularization.

One thing dropout does is that it eliminates hidden units of those layer which has higher chance of overfitting. For this example, layer 1 and 2 has the higher chance of overfitting. And so, dropout will lessen the value of keep-prob of these layers so that some units can be eliminated and chance of overfitting will be less. But for the layer before the output layer, keep-prob remains highest as all the nodes at that point carries important value.[21]
There are some other techniques for regularization or solving overfitting problem.

3.6 Data Augmentation

While training a neural network, use of more data makes it better to solve overfitting problem. But using more data is expensive. To escape from expense, we can modify a less number to produce more data. For an example, for predicting car picture, we can import some pictures of cars to train the model. And we can horizontally flip the pictures, random rotation or crop out some portion of picture and use them as input data. Thus, we can get 3-4 times bigger data set than the number of data that we imported. In this way, we can get a bigger data set without any expense except for some computation cost. Thus, by training a good amount of data we can solve overfitting problem.

3.7 Early Stopping

For overfitting problem, if we plot the cost function of training set, we will see that it going down and down with each iteration. Plotting the development set error, first we will see it goes down to a point and after that it goes higher which leads to

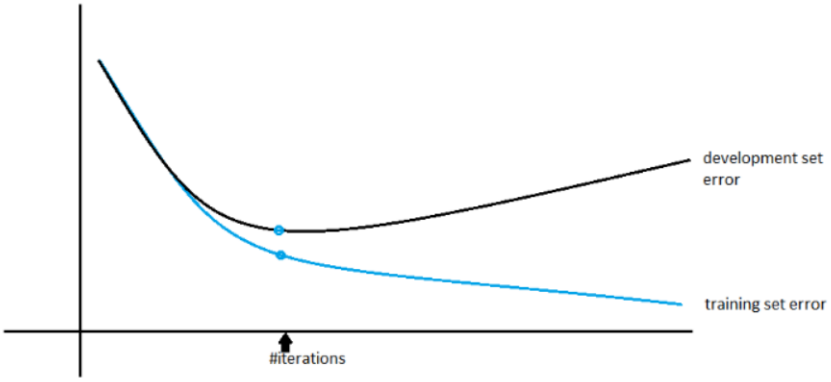

Figure 3.3: Error plot of training and development set

overfitting problem. What early stopping does is that it finds the iteration where there is lower value of development set error and stop the iteration process after that iteration. As the iteration is stopped there, so it can't go up anymore and this reduce overfitting problem. Moreover, at the beginning the value of W$\approx$ 0 as we randomly initialize it to smaller value. And at the end, the value of W gets larger. If we stop at the early stopping point then we get a mid-sized value of W. and in neural network, smaller value of norm W keeps the network away from overfitting. Using these techniques of regularization, we can solve the overfitting problem of a neural network.

3.8 Normalization

To enhance the performance of a neural network, we often use normalization. It normalizes the data of a neural network. While taking data, the input features can

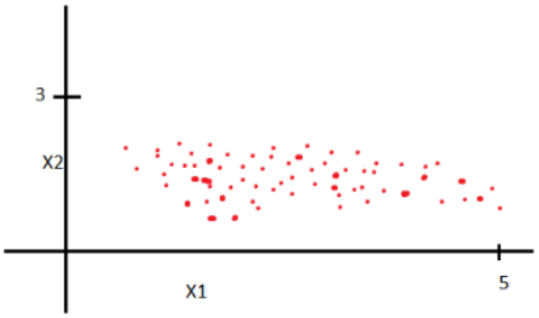

Figure 3.4: Data without normalizing

be from different scale which will lead to W_1 and W_2 being very different range of value. This will make the cost function plot very shorten. But normalizing the data will make the cost function more symmetric.

Normalization is done in two steps. First, we have to zero out the mean. It will take the values closer to zero.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{[i]} \tag{3.12}$$

$$x := x - \mu \tag{3.13}$$

So, it will shift all the value closer to zero. But the values will have high variance as they will be horizontally scattered. To reduce variance:

$$\sigma = \frac{1}{m}\sum_{i=1}^{m} x^{[i]} * *2 \tag{3.14}$$

$$x/ = \sigma^2 \tag{3.15}$$

This will lessen the variance and all the values will be compacted and closer to zero. This way, the cost function will be easier and faster to optimize as all the features will be in same scale.

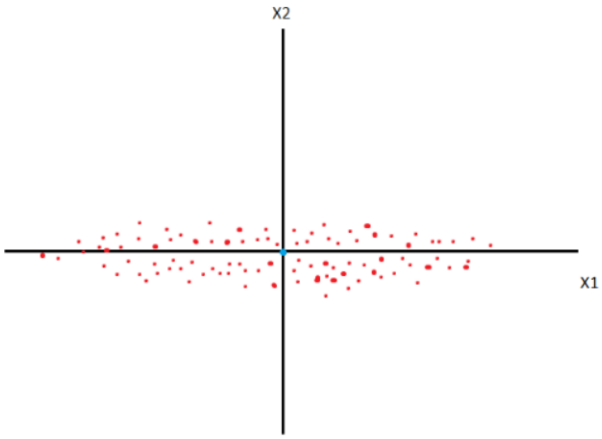

Figure 3.5: Data after making zero mean

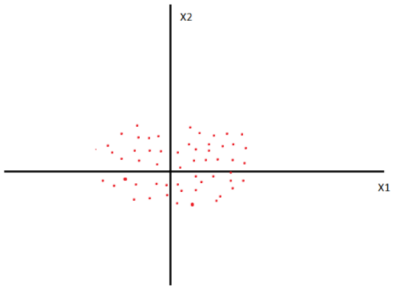

Figure 3.6: Data after zero mean and reduced variance i.e. normalization

This is only in terms of input vectors. But for deeper neural network, beside normalizing input vector X or $A^{[0]}$, we need to normalize other layers input vector $A^{[1]}$, $A^{[2]}$, $A^{[3]}$,....., $A^{[l]}$. This Is known as batch normalization. To normalize $A^{[l]}$, we need to normalize $Z^{[l]}$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} Z^{[l][i]} \tag{3.16}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} Z^{[l][i]} - \mu)^2 \tag{3.17}$$

$$Z^{[l][i]}{}_{norm} = \frac{Z^{[l][i]} - \mu}{\sqrt{\sigma^2 + E}} \tag{3.18}$$

$$Z^{[l][i]} = \gamma Z^{[l][i]}{}_{norm} + \beta \tag{3.19}$$

This E is added as σ might be closer to zero.γ and β are learnable parameters which will be updated along with weight update in each iteration. So, instead of$Z^{[l][i]}$, we will use $Z^{[l][i]}$. after computing $Z^{[l][i]}$ we will normalize it to $Z^{[l][i]}$ and using this we will calculate activation function.

So, the whole process is:

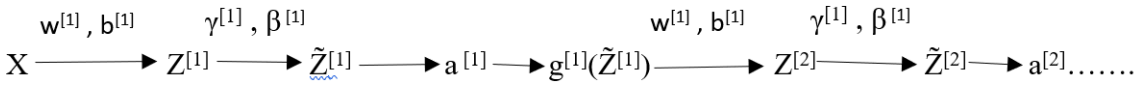

Figure 3.7: Normalization process

This learnable parameter's value is updated like weight. This normalization is applied to mini batch. While implementing it to mini batch, $z^{[l]} = W^{[l]}\ a^{(l-\ 1)} + b^{[l]}$ for this calculation, mean variance subtraction always subtract out $b^{[l]}$. So, for batch normalization in mini batch, we will not use $b^{[l]}$ or $db^{[l]}$.

Batch normalization normalize the $z^{[l]}$ for all layers in same scale. This is for training examples. For testing, we need to process one single example at a time. Using this normalization, we can get a faster and efficient neural network.

3.9 Vanishing/Exploding Gradient

One of the problems while building a neural network is vanishing/ exploding gradient. It means that while training a very deep network, the slopes can be very big

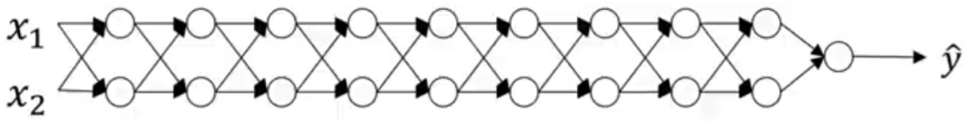

Figure 3.8: Deep neural network

or very small. This makes the neural network work very slowly. For an example, suppose we take a very deep network where g(z)=z and b=0.

Then

$$y' = w^{[1]}\ w^{[2]}\ w^{[3]}\ w^{[4]}.....w^{[L]}\ x \tag{3.20}$$

$$= w^{[1]}\ a^{[1-1]}x; as[a^{[L]} = g(z^{[L]}) = w^{[L]}x \tag{3.21}$$

$$= w^{[L]}\ w^{[L-1]}x \tag{3.22}$$

Here, if the value of W is greater than 1 and value of L is very large (a very deep network) then the value of y' will exponentially increase or explode. And if the value of W is less than 1 then the value of y' will exponentially decrease or vanish. This will make the activation smaller as well as the gradient too. So, gradient will take a very small step in each iteration which will make the neural network train very slowly.

To solve this problem, we need to initialize the weights very carefully. If we take larger number of input then w will be smaller. As Z is the summation of w x, so the

w has to be smaller for that. So, one thing we can do is set the variance (w) = $\frac{1}{n}$

$$w^{[L]} = np.random.randn(shape) * np.sqrt(1/n^{[L-1]}) \tag{3.23}$$

This is for tanh function. For ReLU function the function below works better.

$$w^{[L]} = np.random.randn(shape) * np.sqrt(2/n^{[L-1]}) \tag{3.24}$$

Using this, we can make the value of w in reasonable scale and make the activation function not too big or not too small. This solves the problem with vanishing / exploding gradient.

3.10 Gradient Checking

While doing back propagation, we can use gradient checking to see if there is an error. This helps to debug the back propagation calculation. To use gradient checking, we need to take all the $w^{[1]}$ $b^{[1]}$..... $w^{[L]}$ $b^{[L]}$ and reshape all of them into a big vector Θ.

$$J(w^{[1]}\ b^{[1]}..... w^{[L]}\ b^{[L]}) = j(\Theta) \tag{3.25}$$

And also taking all the $dw^{[1]}$ $db^{[1]}$..... $dw^{[L]}$ $db^{[L]}$) and reshape all of them into a vector Θ.
So, for multiple iteration, $j(\Theta) = j(\Theta_1, \Theta_2, \Theta_3, \ldots)$. For each iteration we need to compute,

$$d\Theta_{aprox}{}^{[i]} = \frac{j(\Theta 1, \Theta 2, ..., \Theta 3 + E)}{2E} \tag{3.26}$$

$$\approx d\Theta[i] \tag{3.27}$$

So, we will have these two vector, $d\Theta_{aprox}$ and $d\Theta$ and we have to check if $d\Theta_{aprox} \approx d\Theta$.
Here, the value of Θ should be 10^{-7}. After calculating this, if we get the result as 10^{-7} then the back propagation computation is good. But if we get bigger value than this like 10^{-5} or 10^{-3} , it means there are some bug in back propagation calculation. Either some value of parameter has become very large or small. If a value of an iteration gives odd value then we have to check the value of $db^{[i]}$ and $dx^{[i]}$ of that iteration, the problem might be in there. We have to perform this for regularization too, except for dropout as it randomly eliminates different units.

This gradient checking makes neural network work slower. Still it is efficient to use this to make sure the computation process is going well. But we should never use this while training as it will be very much slow. We will only use it for debugging. Thus, we can find out error in back propagation calculation using gradient checking.

3.11 Optimization

3.11.1 Mini batch

Optimization helps a neural network work faster. Implementing a neural network always deals with a big dataset. It helps to increase the efficiency of the neural network but it takes a lot of time. Use of optimization helps to make the network much efficient by making it faster. While doing vectorization, we used X and Y vectors to stack up all the x's and y's. But for larger amount of data this process is slower too.

For implementing optimization, suppose we are running gradient decent and working with m training examples where m=1,000,000. Applying vectorization will not be fast. So, we will divide the whole input batch X of m training example into small minibatches. Suppose each mini batch contains 2000 training examples.so we'll have 5000 mini batches. Same thing will be applied for Y too.

$$X = [x^{[1]}x^{[2]}x^{[3]}.....x^{[2001]}x^{[2002]}x^{[2003]}.....x^{[1]}] \tag{3.28}$$

$$X = [x^{\{1\}}x^{\{2\}}....x^{\{m\}}\ ] \tag{3.29}$$

$$Y = [y^{[1]}y^{[2]}y^{[3]}.....y^{[2001]}y^{[2002]}y^{[2003]}.....y^{[1]}] \tag{3.30}$$

$$Y = [y^{\{1\}}y^{\{2\}}....y^{\{m\}}\ ] \tag{3.31}$$

We have to go though the forward and back propagation for every mini batches. Here instead of using X we will use mini batch $x^{[t]}$.

t=1 to 5000

#forward propagation from layer 1 to l by vectorizing using $x^{\{t\}}$ and $y^{\{t\}}$
$Z^{[l]} = W^{[l]}\ x^{\{t\}} + b^{[l]}$
$A^{[l]} = g^{[l]}\ (Z^{[l]})$
$J^{\{t\}} = \frac{1}{2000} \sum_{i=1}^{m}\ L\ (a^{(i)}, y^{(i)}) + \frac{\lambda}{2*2000}\ ||w^{[t]}||_F{}^2$

#back propagation from layer 1 to l by vectorizing using using $x^{\{t\}}$ and $y^{\{t\}}$
$dz^{[l]} = dA^{[l]} * g^{[l]}(Z^{[l]})$
$dW^{[l]} = \frac{1}{m}\ dZ^{[l]}\ A^{[\ l-1]T}$
$db^{[l]} = \frac{1}{m}$ np.sum ($dZ^{[l]}$, axis =1, keepdims =True)
$dA^{[l-1]} = W^{[l]T}\ dZ^{[l]}$
$W^{[l]} := W^{[l]} - \alpha\ dW^{[l]}$
$b^{[l]} := b^{[l]} - \alpha\ db^{[l]}$
This is called 1 epoch. Each epoch travels through the network once. For this case, there will be 5000 epochs. By using this process mini batch, the neural network will run much faster in case of bigger dataset.

In batch gradient, if we plot the cost junction, we find it going down. But in case of mini batches, plotting cost function, $j^{\{t\}}$ will give us a noisy output going downwards. The reason for noise is that some batch can be easy going so the cost of those will be lower. And, some batches can be hard so cost of those will be higher.

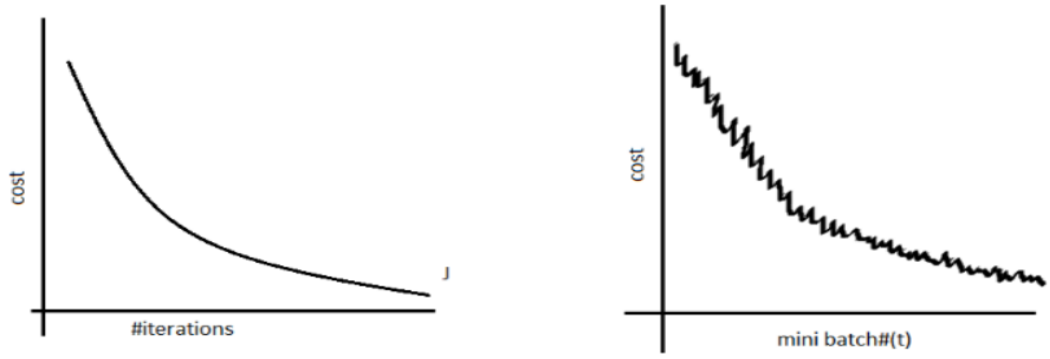

Figure 3.9: Cost function without and with mini batch

It is very important choosing size of mini batch as the performance depends on it. If we take mini batch of size m then it means that it carries the whole input set. It is known as batch gradient decent. For optimizing the cost function, it will take larger steps towards it. On the other hand, if we take mini batch of size 1 then it means that it only carries one training example in each batch. It is known as stochastic gradient decent. For optimizing the cost function, it will be noisy. Sometimes it might go to wrong direction. It will never converge.[23]

So, leaving these two extremes, we should choose a size in between 1 and m. It means that it should not be too small or too big. In this way, vectorization will be fast. In general, size of the mini batches should be 64 to 512 (64,128,256,512) which are power of two like computer memory. It works better this way. This should always fit in CPU or GPU memory. This way, the network works faster. But in case of smaller training set, we should always use the batch gradient decent.

3.11.2 Exponentially Weighted Moving Average

It is another way for optimization. Here, we make the average taking each input one by one. The equation for this is

$$V_t = \beta V_t + (1 - \beta)\Theta_t \tag{3.32}$$

If we calculate for t training examples, the calculation will be,

$$V_\Theta := 0 \tag{3.33}$$

$$V_\Theta = \beta V + (1 - \beta)\Theta_1 \tag{3.34}$$

$$V_\Theta = \beta V + (1 - \beta)\Theta_2 \tag{3.35}$$

$$\vdots$$

$$V_\Theta = \beta V_\Theta + (1 - \beta)\Theta_t \tag{3.36}$$

By changing the value of β ,we can get the number of inputs average it is showing.To

determine that we use $\frac{1}{1-\beta}$. If we use smaller value of , the weighted average curve will be noisy. With the increasing value of , the curve will get smother.

But in the initial stage, this does not give good result as it just starts to warm up. To get a good result in the initial stage we need to do bias correction. Using the bias correction, the equation will look like this for t-th training example:

$$V_t = \beta V_t + (1-\beta)\Theta_t \tag{3.37}$$

$$V_t = V_t/(1-\beta^t) \tag{3.38}$$

When the value of t is smaller, β^t will be bigger. So bias correction will affect V_t a lot. But as we approach to larger value of t, β^t will get smaller, even close to zero. Then bias correction will have no effect. This is why it is used to correct the initial values.

3.11.3 Momentum

There is another algorithm known as momentum. It works faster than gradient decent. If we compute the cost function plot of gradient decent with mini batch, we will see that it takes an oscillated path towards the minimum cost. But for efficiency, it not good to take bigger steps on vertical axis as we don't want faster learning there. On the other hand, we want to take bigger steps towards horizontal axis so that we can get faster learning there.

To implement momentum, we need to compute dw and db on t iterations of mini batch using exponentially weighted average.

$$V_{dw} = \beta_1 V_{dw} + (1-\beta_1)dw \tag{3.39}$$

$$V_{db} = \beta_1 V_{db} + (1-\beta_1)db \tag{3.40}$$

$$W := W \ - \ \alpha V_{dW} \tag{3.41}$$

$$b := b \ - \ \alpha V_{db} \tag{3.42}$$

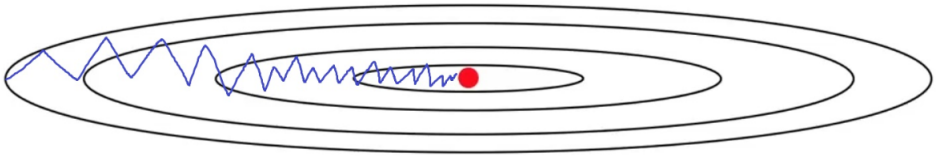

Figure 3.10: Computing Cost Function

When we make the average, the vertical axis's values are averaged out, closer to zero. So, this eliminates the oscillation making the path smoother. And around the horizontal axis, the average value tends to go towards minimum cost in bigger steps. So, it makes learning faster.

Here, it has 2 hyperparameters α and . We usually use 0.9 as value of . So, $\frac{1}{1-} = \frac{1}{1-\ 0.9} = 10$, it shows average value of 10 gradients. As using this, it is warmed up well initially taking average of good numbers of gradients, so we don't need to use bias correction for momentum.[24] Using momentum, we can get faster result than using usual gradient decent.

3.11.4 RMSprop

As we know, to get to the minimum cost, we want fast learning towards horizontal axis and slow learning towards vertical axis by making the oscillation smoother. To do that, there is another algorithm called RMSprop. To implement this, we need to compute dw and db on t iterations using exponentially weighted average.

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dw^2 \tag{3.43}$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2 \tag{3.44}$$

$$W := W \ - \ \alpha \frac{dw}{\sqrt{Sdw} + E} \tag{3.45}$$

$$b := b \ - \ \alpha \frac{db}{\sqrt{Sdb}} \tag{3.46}$$

The dw^2 and db^2 will be calculated elementwise. Squaring the dw will make the value smaller and so S_{dw} will be smaller. While calculating W, dividing with root of a smaller value will make $\frac{dw}{\sqrt{Sdw}}$ larger making W smaller. In this case, value of S_{dw} is closed to zero which can leads to exploding gradient. To stop that, E , a small value is added with S_{dw}. On the other hand, Squaring the db will make the value larger and so S_{db} will be larger. While calculating b, dividing with root of a larger value will make $\frac{dw}{\sqrt{Sdb}}$ smaller making b larger. So, the learning in vertical axis will be slower and learning in horizontal axis will be faster.

3.11.5 Adam

Adam(Adapted moment estimation) optimizer has been doing very well for deep neural network. It is the most used optimizer now a days. Adam optimizer combines momentum and RMSprop. For this we need to initialize some variance to zero.

$$V_{dw} = 0, S_{dw} = 0, V_{db} = 0, S_{db} = 0 \tag{3.47}$$

We need to compute dw and db on t iterations of mini batch using exponentially weighted average.

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) dw \tag{3.48}$$

$$V_{db} = \beta_1 V_{db} + (1 - \beta_1) db \tag{3.49}$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) dw^2 \tag{3.50}$$

$$S_{db} = \beta_2 S_{db} + (1 - \beta_2) db^2 \tag{3.51}$$

The we need to perform bias correction.

$$V_{dw}{}^{corrected} = V_{dw}/(1-\beta^t{}_1) \tag{3.52}$$
$$V_{db}{}^{corrected} = V_{db}/(1-\beta^t{}_1) \tag{3.53}$$
$$S_{dw}{}^{corrected} = S_{dw}/(1-\beta^t{}_2) \tag{3.54}$$
$$S_{dw}{}^{corrected} = S_{dw}/(1-\beta^t{}_2) \tag{3.55}$$

$$W := W - \alpha(V_{dw}{}^{corrected}/\sqrt{Sdw}^{corrected} + E) \tag{3.56}$$
$$b := b - \alpha(V_{dw}{}^{corrected}/\sqrt{Sdw}^{corrected} + E) \tag{3.57}$$

The default values of hyperparameters are $_1$=0.9 , $_2$=0.999 and E=10^{-8}. This adam optimizer has been proved as the most efficient one as it performs well with any structure or size of neural network.

3.12 Learning Rate

The value of learning rate effects a lot in algorithms. While we work with mini batch, if we keep constant larger value of learning rate, initially it will work faster. But going in towards minimum cost, it goes in wrong direction. This is why it never converges. Rather than converging, it wanders around.

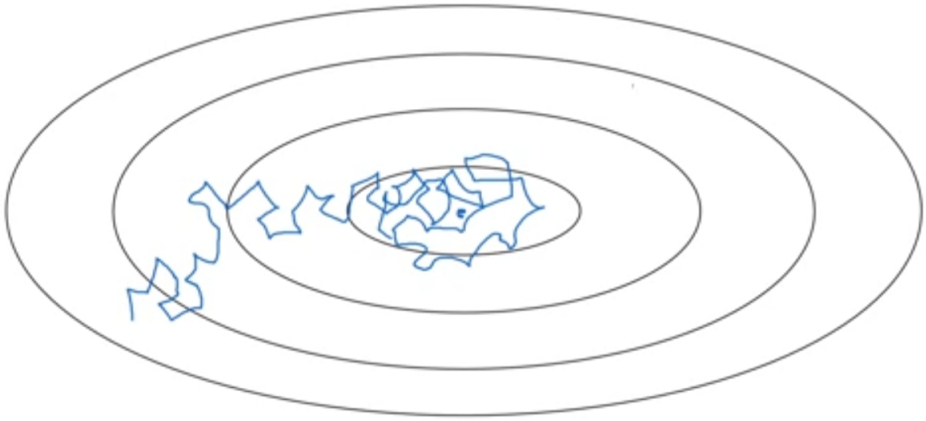

Figure 3.11: Converging with higher learning rate

Rather than using a constant value of learning rate, if we reduce the value gradually, it will work better. The learning rate reducing process is called learning rate decay. At first it takes larger steps to make the learning faster. But later on, it takes little steps and wanders around very closer to minimum cost. Though it never converges, it stays closer to minimum cost.

The equation for learning rate,

$$\alpha = \frac{1}{1 + decay\ rate * epoch\ number}\ \alpha_0 \tag{3.58}$$

Here, epoch numbers=1,2,3,. . . . . ,t

With the rise of epoch number value learning rate decreases as it gets divided by a higher number gradually.

There are some other equations that can be used for learning rate.

$$\alpha = 0.95^{epoch\ num}\ \alpha_0\ ; \tag{3.59}$$

This exponentially decreases learning rate's value.

$$\alpha = \frac{k}{\sqrt{epoch\ number}}\ \alpha_0 \tag{3.60}$$

$$\alpha = \frac{k}{\sqrt{minibatch}}\ \alpha_0 \tag{3.61}$$

we can perform manual decay too. Choosing a good value of learning rate can really make the algorithm run faster.

3.13 Softmax Regression

Previously, we have seen that we can do prediction of a single class using neural network. It means that from a picture, we can identify whether it is a car or not. This is classifying into one category or one class. But using neural network, we can also classify into multiple categories or classes. Such as, from a picture, we can identify whether it is car or bus or truck or neither. For this, we use softmax regression.

C=number of classes or categories

For this, in the output layer number of units will be $n^{[l]}$=C
For the output layer,

$$Z^{[l]} = W^{[l]}\ A^{[l\text{-}1]} + b^{[l]} \tag{3.62}$$

Activation function:

$$t = e^{z^{[l]}} \tag{3.63}$$

$$a_i^{[l]} = \frac{t_i}{\sum_{j=1}^{c} t_i} \tag{3.64}$$

$$a^{[l]} = g^{[l]}\ (z^{[l]}) \tag{3.65}$$

Using this, we'll get the probability of each unit of output layer which is categorized into proper class.

For example,

$$\text{for } z^{[l]} = \begin{bmatrix} 5 \\ 6 \\ -1 \end{bmatrix}$$

$$t = \begin{bmatrix} e^5 \\ e^6 \\ e^{-1} \end{bmatrix}$$

$$a^{[l]} = g^{[l]}\,(z^{[l]}) \tag{3.66}$$

$$= \begin{bmatrix} e^5/(e^5+e^6+e^{-1}) \\ e^6/(e^5+e^6+e^{-1}) \\ e^{-1}/(e^5+e^6+e^{-1}) \end{bmatrix} = \begin{bmatrix} 0.268 \\ 0.730 \\ 0.001 \end{bmatrix}$$

This is the probabilities of each class. The calculation of loss function for softmax will be:

$$L(y`, y) = -\sum_{j=1}^{c} y_j log y`_j \tag{3.67}$$

and cost function,

$$J(w, b) = \frac{1}{m}\sum_{i=1}^{m} L(y`^i, y^i) \tag{3.68}$$

This is for forward propagation. For back propagation,

$$dz^{[i]} = y` - y \tag{3.69}$$

By following these, we can implement softmax on a neural network for multiple classification.

Chapter 4

Comparison of different Models of Neural Network

4.1 Recurrent Neural Network (RNN)

Like the other models of deep neural networks, RNN is also comparatively old. RNN was introduced and design in 1980's but at that time people couldn't utilize properly this architecture. In recent years because of the ability of computational power and the huge amount of data, RNN is showing its true potential and power.

Recurrent Neural Networks (RNN) are a strong and robust kind of neural networks. They belong to the foremost promising algorithms out there at the present time as a result of they're the sole ones with an internal memory. As RNN has the ability to remember the necessary information about what happened in the recent past combining with input it can predict properly what will happen next. So we can say recurrent neural networks are that type of network that can capture information from sequence or time series data and guess the next phenomenon.

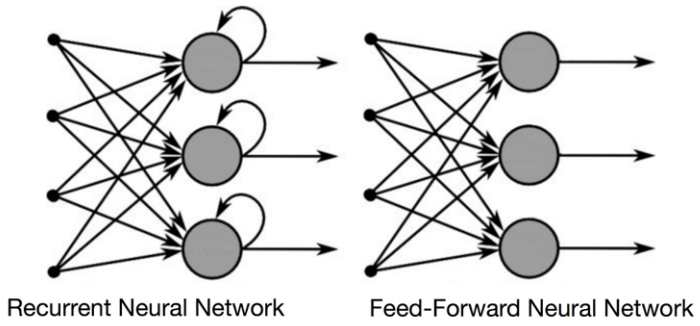

Figure 4.1: Difference between RNN and feed forward network

Typical feed forward neural network model doesn't have any memory of what happened before. Here data move from input layer, through hidden layer, to output layer in one direction. As they give prediction only based on the current data their prediction for sequence data is really bad where future information depends on the past information. Therefore, traditional neural networks don't have this kind of ability to make deeper understanding of sequence and its context. As a result, for

sequential data like time series, speech recognition, text, financial data, language modeling, translation, image captioning, audio, video, weather and much more we tend to choose this RNN algorithm because we need past information to predict and RNN loops the information back to the node.[26]

For example, if a network is analysis a sentence suppose the input is "be kind to others". Here network is analyzing it word by word. So when it is time to predict the word "others", feed forward network doesn't have past information to analyze. So here comes RNN to solve this issue.

4.1.1 How RNN works

In a simple RNN, it loops back it information in memory. So it actually has two input one is current state input and another is past information. But combining both information it produces its output. Following is figure of simple RNN.

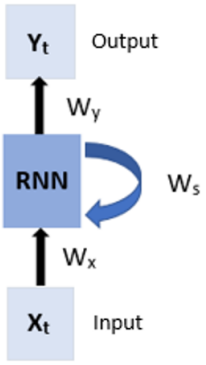

Figure 4.2: Structure of a simple RNN

RNN basically works on a basic recursive formula like this,

$$S_t = F_w(S_{t-1}, X_t) \tag{4.1}$$

Here, X_t = input at time step t
S_t = state at time step t F_w = recursive function

The new state (S_t) of recurrent neural network at time t is a function of its old state (S_{t-1}) at time t-1 and the input (X_t) at time t. Here the following figure is the basic structure of RNN where,

$$S_t = tanh(W_s S_{t-1} + W_t X_t), \tag{4.2}$$

tanh is the recursive function.

$$Y_t = W_y S_t \tag{4.3}$$

RNN take current input X_t and previous state S_{t-1} and multiply them with their appointed weight and create new state S_t. from this new state it gives the output.

Following is unrolled version of recurrent neural network.

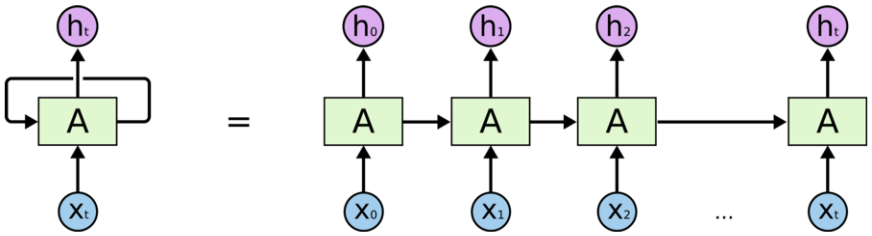

Figure 4.3: An unrolled RNN

As we can see after unrolling there is no cycle because in unrolled version we can visualize different timestep and information passed on the following timestep.

4.1.2 Problems with RNN

1. Vanishing Gradient Problem:
While training neural network we use backpropagation to find out the local minimum which is done by taking repetitive tiny steps. If the gradients are too small it takes much more time to learn because while moving one layer to another gradients become exponentially small and vanishes at the end meaning it becomes close to zero when it comes back to the initial layers. Because while gradients are being propagated back, in the deep layers they have to undergo repetitive multiplication which make it smaller in every step. So, learning becomes very much slower and weights take too much time to update making the network inefficient.

2. Long term Dependency:

One of the most important features of RNN is the knowledge of past and ability to make connection of it with present task. Sometimes we need recent past information to guess next or sometimes we need to know more information about what happened long ago.[27]

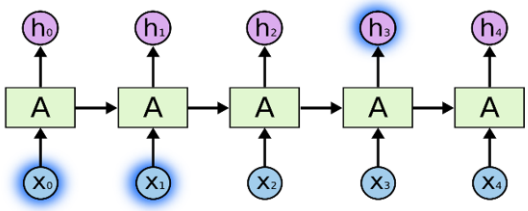

Figure 4.4: Small gap between connected information

For example, to predict the last word in a sentence like this "Running every day is good for *health*". To know the last word "health" we don't need much context of what happened earlier it is quite obvious.

But for a story like this "Though I born in Chandpur I grew up in Chittagong...I also knew the local language of *Chittagong*", we need the earliest context to be able

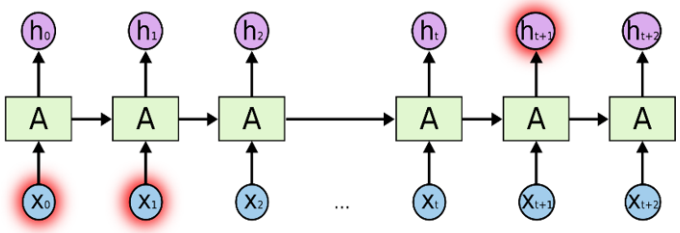

Figure 4.5: Large gap between connected information

to narrow down which language he is talking about. So the interval between a data and where this information will be needed can vary. In theory RNN is able to do that but practically when this gap becomes too huge RNN becomes incapable of connecting them meaning RNN model cannot learn properly

This is where LSTM comes to solve these problems.[28]

4.2 LSTM (Long Short Term Memory) Networks

Basically long short term memory network is an extended version of RNN which makes it capable of expanding their memory. Therefore it can learn from long term experience from their extended memory making them suitable for problem where the gap between relevant information is large.LSTM is capable of remembering much previous context and connect it to recent data because it can read, write and delete data in its cell as necessary. It can manipulate which data it need not to remember now and which is important for this new state by analyzing previous and current information. LSTM can do this because it has some gates to control the information flow.[29] Following is the figure of normal LSTM cell.

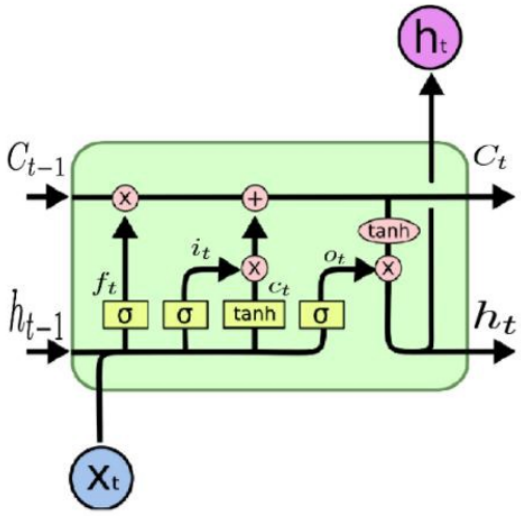

Figure 4.6: LSTM cell

There are 3 gate in a LSTM cell. They are forget gate, input gate and output gate. In the forget gate layer LSTM decides which information is important to remember

and which is not. It looks at the input X_t and previous state h_{t-1} and give output of 0 to 1 because it used sigmoid function. We can write it as,

$$f_t = \sigma(W_f . [h_{t-1},\ x_t] + b_f) \tag{4.4}$$

Nest step of LSTM is going to be what information it needs to store. It is combination of 2 step. In input gate layer it calculates the input X_t which determine what values should be update and intermediate cell state $\tilde{C}_t$ which consider the new candidates for the next state. By combining this two with the forget layers output we will get a new cell state. The equations are following,

$$i_t = \sigma(W_i\ .[h_{t-1}, x_t\ ] + b_i) \tag{4.5}$$

$$\tilde{C}_t = \tanh(W_C . [h_{t-1}, x_t] + b_C) \tag{4.6}$$

$$C_t = (f_t *\ C_{t-1}) + (i_t *\ \tilde{C}_t) \tag{4.7}$$

By multiplying old state with forget layer output LSTM decide what to forget and combine with intermediate state we get new state of cell. Now LSTM will give output on the basis of a filtered version of cell state. It will run a sigmoid on the input, later which will decide from what part of cell state the output will be. And then by multiplying o_t with the tanh function of new cell state C_t we get the final output h_t.

$$o_t = \sigma(W_o\ .[h_{t-1}, x_t] + b_o) \tag{4.8}$$

$$h_t =\ o_t * \tanh(C_t) \tag{4.9}$$

This is how a simple LSTM works but there are lots of variants on LSTM.

4.3 Convolutional Neural Network:

CNN is another special type of neural network which is used for processing data when it has grid like topology. This data can be one dimensional time series which is grid of samples over time or something like even two-dimensional image data which is grid of pixels in space. Mostly in recent years convolutional neural network has gain much popularity because of its capability to analyze image and classify them. CNN is very much useful in image data because the convolutional layers in this architecture can pick out or detects pattern in images and make sense of them.[30]

The layers in a convolutional neural network can be classified in 4 layers which are convolutional layer, activation layer, pooling layer, fully connected layer. Here in the first layer in a CNN would be convolutional layer where it can convolve the image or data in general using filter. This is what makes CNN different from MLP where by using filter in convolutional layer CNN can detect patterns in image like shapes, edges, curves, objects, colors, textures etc. Now filters are small unit that we apply across the data through a sliding window. The convolutional operation involves taking the dot product of filters with same size portion of the image and then summing those values for every sliding action. In the first convLayer it perceives low level feature of the image gradually in the deep layers it captures high level feature and help us to have an overall assumption about image.

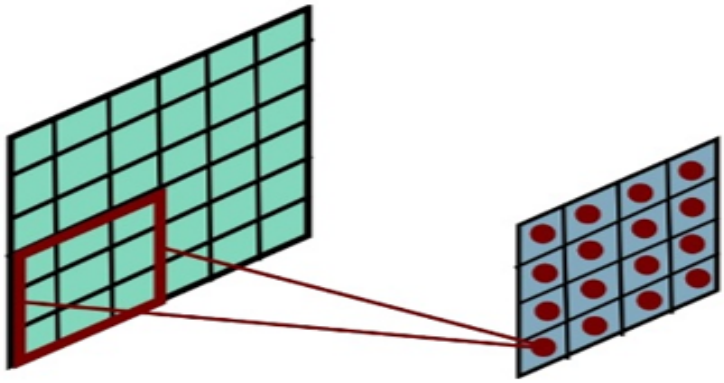

Figure 4.7: Convolution Layer

In the next activation layer a nonlinear activation function is used otherwise it won't be a learning if the function is linear. After that comes the pooling layer which is used to reduce the spatial size of convolutional feature in a network to have efficient learning and faster convergence time. By reducing dimensionality we need less computational time which leads to faster learning and less parameter to remember. Max polling and the average pooling are the two types of pooling. Max pooling compute the max of a portion of image on the other hand average pooling compute the average.

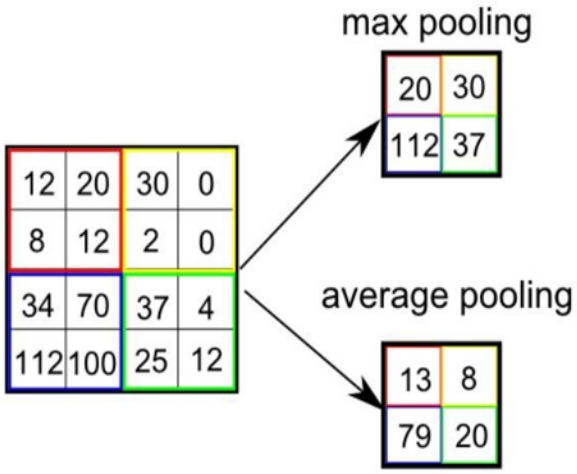

Figure 4.8: Max Pooling and Average Pooling

The output of convolutional layer represents high level feature of images and a cheap way to learn nonlinear combinations of this data is adding a fully connected layer as the output of pooling layer is 3D map of image we need to flatten the data into a column vector to feed it in the fully connected layer where it can learn nonlinear combination through many epochs. And in the final output layer by using a softmax classification we can show the output.

Following is a sample figure of CNN,

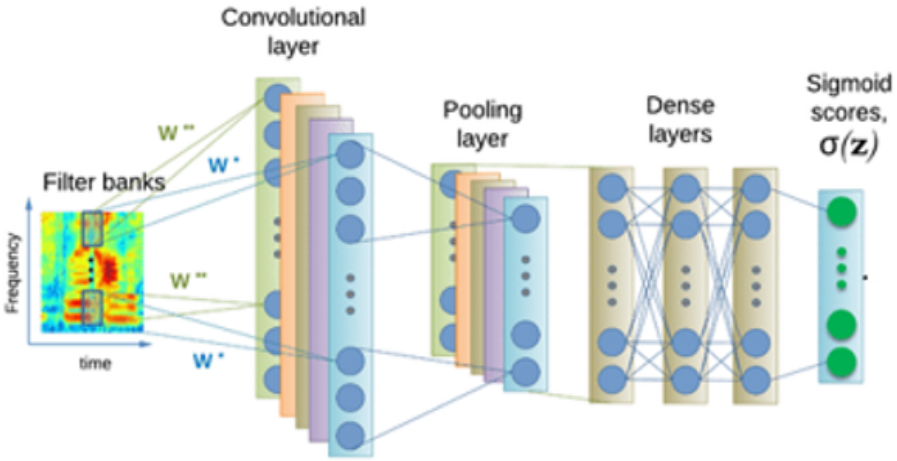

Figure 4.9: Architecture of CNN

Chapter 5

Application of Neural Network in Finance

We have seen in the previous chapters how we can build a neural network, how can we make it efficient and some learning algorithms of neural network. Its efficiency and fast converging have made it very successful for any kind of implementations. Use of neural network has taken over many fields of science. There are a lot of applications where neural network has been used. They are:

1. Medical diagnosis
2. Financial forecasting
3. Process modeling and control
4. Machine diagnostics
5. Target recognition
6. Portfolio management
7. Credit rating
8. Targeted marketing
9. Voice recognition
10. Intelligent searching
11. Fraud detection
12. Image compression
13. Character recognition

Among them, in this paper, we will work with financial forecasting using neural network. Now a day's financial world is very complicated. Many factors weigh in to make a change any moment. This is very hard to handle. But use of neural network has been helping to predict or manage a lot of areas of finance. They are:

1. Stock market prediction

2. Currency prediction
3. Bond ratings
4. Business failure prediction
5. Debt risk assessment
6. Credit approval
7. Bank theft
8. Bank failure

In this paper, we will focus on how we can predict stock market using neural network.

5.1 What is Stock Market

Stock market is a public market where people can issue, buy and sell stocks that trade on a stock exchange. Stock are ownership equity of a company. This activity helps companies raise necessary capital from investors. It means that a company divides itself into a number of shares. Stock market indexes can go up or down which mean that the index might have gained or lost values as a whole. Investors who buy and sell stock use this movement of stock price to make profit.

5.2 Stock Market prediction

Stock market is an important part of economy of a country. It plays a significant role in growth of industry in a country which leads to economic prosperous. Many business and people invest their money on stock market. A single wrong decision can cause them a huge loss. On the other hand, a decision can give them a big profit. For this decision making, it is very important to predict the price of stock market. Many big investors often look for a way to predict stock market. As this is a very complex market, it is not so easy to predict stock market. There are several machine learning methods which have been used to predict stock market. But no methods were found to give 100 percent accuracy. As neural network works efficiently than other methods, we will use neural network to predict stock market in this paper. In the next chapter, the implementation process of neural network in stock market data is given.

5.3 Contributions

In this study our goal was to give the proper idea behind a neural network. Here we studied from the very scratch about how a neural network process the information and how the calculation was done in the nodes of different layers. A feed forward neural network result cannot be good because all the values mostly initialized randomly. So then we need the back propagation to reduce the cost. By using the back propagation method we are actually trying to reduce the gradient to find the global

optimum which will reduce the cost. Here we talked about different types of activation function to apply in neural network model and when and where we should use which type of activation function. Further, we research more about how to improve the efficiency of neural network. So we started to learn about the hyperparameters and optimizations which can upgrade the performance of a neural network architecture. After we inquiry all about this we decided to apply some different types of neural network model one financial problem so that we can figure it out how well a neural network model work on real life problems.

There are various types of financial problem exists today and we decided to forecast the stock price with neural network. We used two different dataset (Amazon and SP 500) to predict the prices so that we can understand if it actually work on different types of nonlinear dataset. So we collected data from yahoo finance which is a popular source of stock price data of various company. We collected and processed the data to use in our models. We used 3 model for this prediction which are MLP, LSTM and CNN. We used the library keras and tensorflow to build the models in python. To train we used mean squared error as loss function and mean absolute error as metrics. Then we fed our train dataset to train the model and test it with test dataset. We used adam as the optimizer. The results we got have given in the later chapter.

As initially we used around 250 days of daily price data we found out result was not that good. So we increased our data to 755 days of daily price. Moreover we added some dropout layer which actually made the prediction of model better. Also, initially we used 50 days data to train and giving the prediction of 51st day. But we reduced it to 25 and got better prediction.

Studying for neural network has been going on for some time. So our focus was to study those different idea and accumulate it and later apply on some real data. We researched about a lots of techniques which can improve neural network and gather them here. In previous work there wasn't a lots of study about SP 500 dataset. Some of the work we found they are mostly worked on popular machine learning algorithm like logistic regression, random forest, support vector machine etc. but we wanted to focus on different neural network models and their performance on stock data. And the results we got was pretty good to say that neural networks model does a great job on nonlinear data where previously they don't have any information on it and even after that they can capture the relationship between data.

For the time shortage we couldn't experiment it with more neural network model. But in future we think of studying more in this field and apply other models like gated recurrent unit, modular neural network, radial basis network etc. moreover, we want to study if we can actually find a pattern to initialize hyperparameter for different architectures so that we don't have to check every time with different value for finding out a proper value for a better output. Moreover we want to study different techniques to integrate with neural network like fuzzy interface, auto encoder, genetic techniques etc. and find out their performance on real life data.

Chapter 6

System Implementation

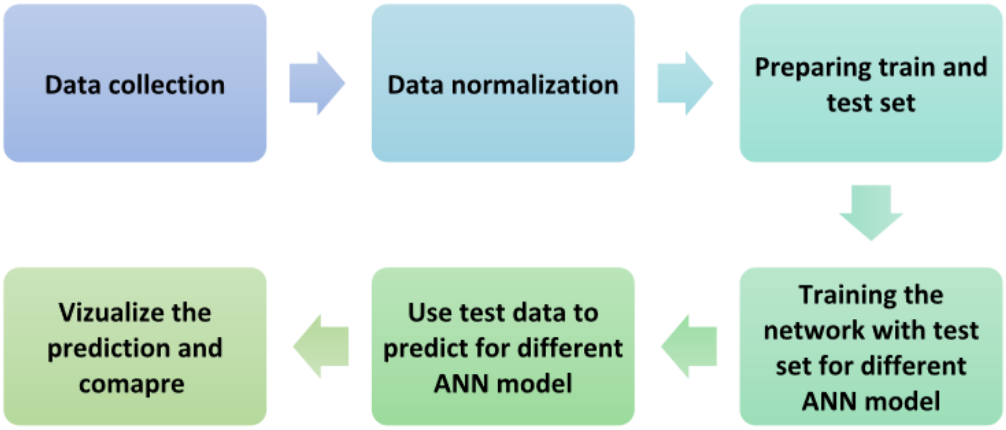

Figure 6.1: System Implementation Process

6.1 Data Collection

Finding the good and related data is one of the important part of predicting model. As we are going to predict the stock price we chose to collect the data from yahoo finance which is a good source for finding stock data for different companies. We will implement our neural network model on 2 different dataset and will see if it can predict the prices in different time step. Therefore, we decide to collect data of amazon and S&P 500 index. To collect data from yahoo finance is not a complex task. As the data is available in their website we can download the csv format from historical data or pull data directly with python API. For both amazon and S&P 500 index we collected data from 2016 to march 2019. Both data contains-

1. Date
2. Open- opening price of the company
3. High- highest price on that day

4. Low-lowest price on that day
5. Close-closing price of that day
6. Adj close
7. Volumn

We will use the feature 'open' column to train and predict the opening value.

```
In [26]: df_amzn = pd.read_csv('AMZN.csv')
    ...: data_set = df_amzn.iloc[:, 1:2].values
    ...: print(df_amzn.head())
         Date        Open        High  ...       Close   Adj Close   Volume
0  2016-03-28  584.400024  584.750000  ...  579.869995  579.869995  3121500
1  2016-03-29  580.150024  595.849976  ...  593.859985  593.859985  4167100
2  2016-03-30  596.710022  603.239990  ...  598.690002  598.690002  3890500
3  2016-03-31  599.280029  600.750000  ...  593.640015  593.640015  2681800
4  2016-04-01  590.489990  599.030029  ...  598.500000  598.500000  2917400
```

Figure 6.2: Data set of Amazon

```
In [27]: df_snp = pd.read_csv('snp.csv')
    ...: data_set = df_snp.iloc[:, 1:2].values
    ...: print(df_snp.head())
         Date         Open         High  ...        Close    Adj Close      Volume
0  2016-03-28  2037.890015  2042.670044  ...  2037.050049  2037.050049  2809090000
1  2016-03-29  2035.750000  2055.909912  ...  2055.010010  2055.010010  3822330000
2  2016-03-30  2058.270020  2072.209961  ...  2063.949951  2063.949951  3590310000
3  2016-03-31  2063.770020  2067.919922  ...  2059.739990  2059.739990  3715280000
4  2016-04-01  2056.620117  2075.070068  ...  2072.780029  2072.780029  3749990000
```

Figure 6.3: Data set of SP 500 index

6.2 Data Scaling

Normalizing the data is another significant part of data processing. We need to scale our data because sometimes the range of data may vary widely and because of that learning can be slow so we are going to squish the data in a range of 0 to 1 using python library.

6.3 Preparing Train and Test Data

After scaling the data we are going to split it to train and test set. We are going to implement 3 model of neural network which are MLP, LSTM, CNN. For MLP and LSTM train dataset contain 70% of total data and for CNN train dataset contains

```
19
20 # data Scaling
21 from sklearn.preprocessing import MinMaxScaler
22 sc = MinMaxScaler(feature_range = (0, 1))
23 data_set_scaled = sc.fit_transform(data_set)
24 #print(len(data_set_scaled))
25
```

Figure 6.4: Data Normalization

80% of total data. For CNN by tweaking the train and test data we found out it can predict better if it has larger train data. Here we are going to predict the next day open price of company by analyzing previous 25 timestep. So if first 0-24 index is input, the predicted output will be the value of 25^{th} index. After splting data, to train our models we create X_train and Y_train array where X_train is the training data and Y_train is the output. In the same way we create X_test and Y_test to test the models.

6.4 Building the Neural Network Models

We are going to build out network by using keras with tensorflow in backend.

1. **MLP:**

   We created simple 3-layer multiple layer perceptron with ReLU activation in each layer. First two hidden layers contain 100 node each and final output layer contain one node.

2. **LSTM:**

   After creating a sequential model, we added 3 LSTM layer in the architecture and an output dense layer. First, second and third LSTM layer has consecutively 256,128 and 64 node and the output layer contains one node. We also added dropout layer to prevent overfitting. We also use ReLU activation here.

3. **CNN:**

   As we are training a sequential time series data we used CONV1D layer of keras as convolutional layers.

   Here we added 2 convolutional layers of 64 node each and then a maxpooling layer. From that output we again feed them into 2 more convolutional layer of 128 node each and then a global average pooling layer. Finally, an output layer of 1 node is added.

6.5 Train the Model

Now we are going to train the model to predict the stock market. We need to define two important term to train the model which are optimizer and cost function.

1. **Cost Function:**

   If a neural network model has done a good job or not with their training set and expected output is measured by cost function. Our goal is to reduce the cost function of model but again we don't want to overfit the model. We used "mean square error" i.e. mse. It can calculate the squre deviation of target and prediction.

2. **Optimizer:** not only we want to reduce cost of our model but also we need a optimizer to invoke the gradient calculation to adjust the weights and biases so that cost function van minimize with every epoch. In these models we used "Adam" optimizer.

We trained our model with batch size of 32 and in 100 epoch.

```
70
71 # Compiling the model
72 regressor.compile(optimizer = 'adam', loss = 'mean_squared_error', metrics=['mean_absolute_error'])
73 # Fitting the model to the Training set
74 history=regressor.fit(X_train, y_train, epochs = 100, batch_size=32)
75
```

Figure 6.5: Training the Model

After training we can see the graph of model loss. For example for long short term memory architecture model loss for amazon and SP 500 index are following.

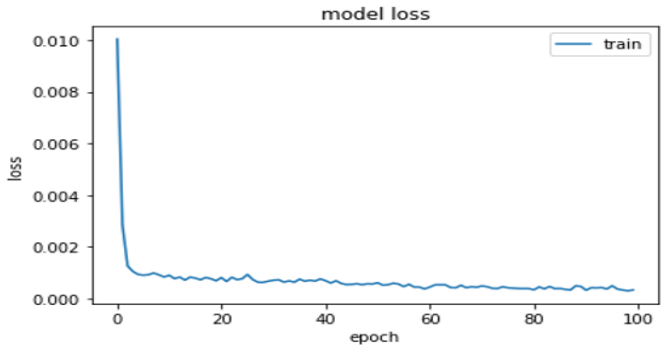

Figure 6.6: Graph of model loss in LSTM of Amazon dataset

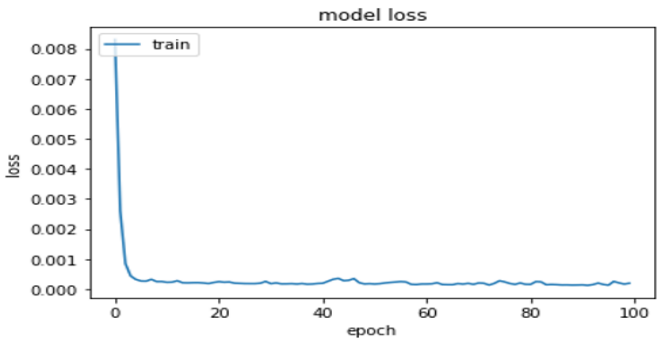

Figure 6.7: Graph of model loss in LSTM of SP 500 index dataset

6.6 Testing the Model

Now the training is done and it is time to test the model to see if it can predict the price of stock market. Result and analysis is shown in next chapter.

Chapter 7

Result and Analysis

Now the training is done and it is time to test the model if it can predict the price of stock market. To predict the stock price, we fed test data into the different model and compare it with the test output to check if the model can do a good prediction or not. As we said earlier, we used two different stock price datasets so that we can find out if the model is capable of adapting with changing values of different sequential data and give a better prediction. We calculated R∧2 value for each model of both datasets to find out how good the model is to forecast stock market price. R∧2 is a measure of how flexible a network is or we can say it actually shows the proportion of variance of the dependent variable which is determined by the independent variable and its relation with the dependent variable. The value of R∧2 is higher if the model can get the idea if variation in dependent variable.

We applied multiple layer perceptron, long short term memory and convolutional neural network for the amazon and S&P 500 stock price data. From the 3 different model we calculated loss (mean squared error), mean absolute error and R∧2 value. From the values of different architecture, we can see convolutional neural network did not done any good prediction and to get an average prediction it needs 20% more data than multiple layer perceptron and long short term memory. Following is the comparison of the models for both companies.

Amazon stock price			
Model	**loss**	**Mean absolute error**	**R∧2**
MLP	1.8619e-04	0.0094	0.9007
LSTM	3.3398e-04	0.0133	0.8854
CNN	0.5127	0.0277	0.6343

S&P 500 index			
Model	**loss**	**Mean absolute error**	**R∧2**
MLP	4.4537e-04	0.0352	0.8682
LSTM	3.2411-04	0.0117	0.9259
CNN	0.5460	0.0642	0.6882

From the above comparison it is quite clear that convolutional neural network is not quite effective to analyze time series data and forecast financial stock market. We used 60% of total data in multiple layer perceptron and long short term memory and was able to get a good prediction. On the other hand, in convolutional neural network if we use 60% data the outcome is very poor. R∧2 value was closer to 0. But when we feed more data in convolutional neural network it started to predict better but that was not enough. It couldn't properly interpret the relationship among variables.

On the contrary, we can see it is possible to get promising result using long short term memory network which is an extended version of recurrent neural network. Both multiple layer perceptron and long short term memory gave good prediction but long short term memory was more consistent.

7.1 Graphical Analysis

For the Amazon Company the comparison between predicted and real stock price for different models are following:

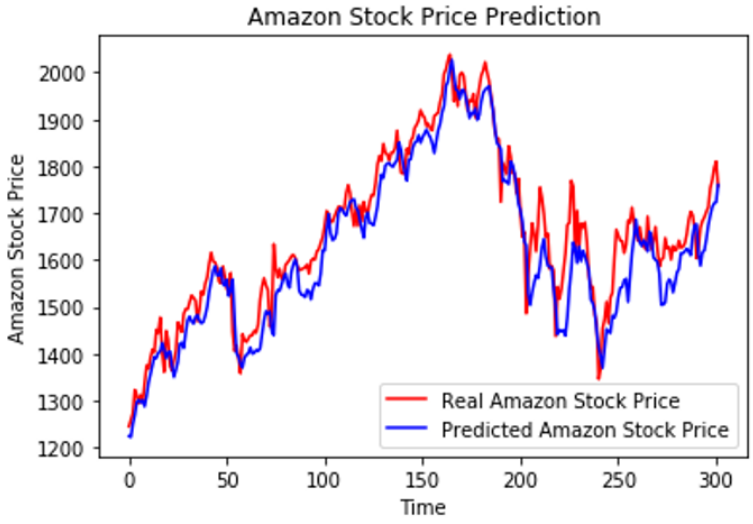

Figure 7.1: Graph of real and predicted price of amazon using multiple layer perceptron (MLP)

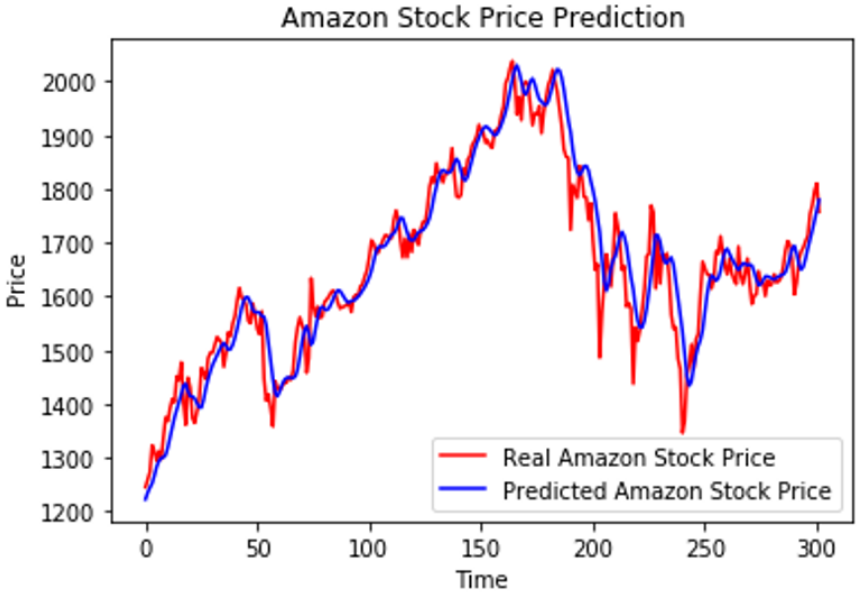

Figure 7.2: Graph of real and predicted price of amazon using long short term memory (LSTM)

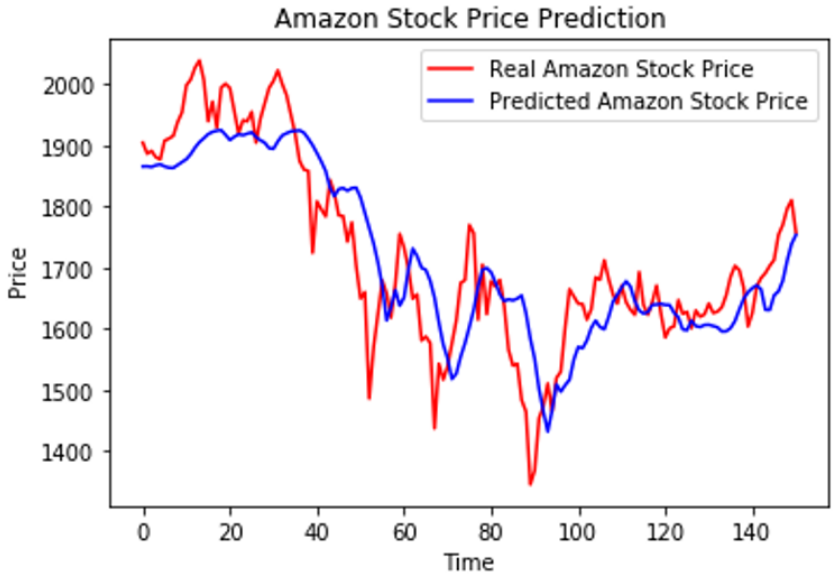

Figure 7.3: Graph of real and predicted price of amazon using long convolutional neural network (CNN)

For the SP 500 index the comparison between predicted and real stock price for different models are following:

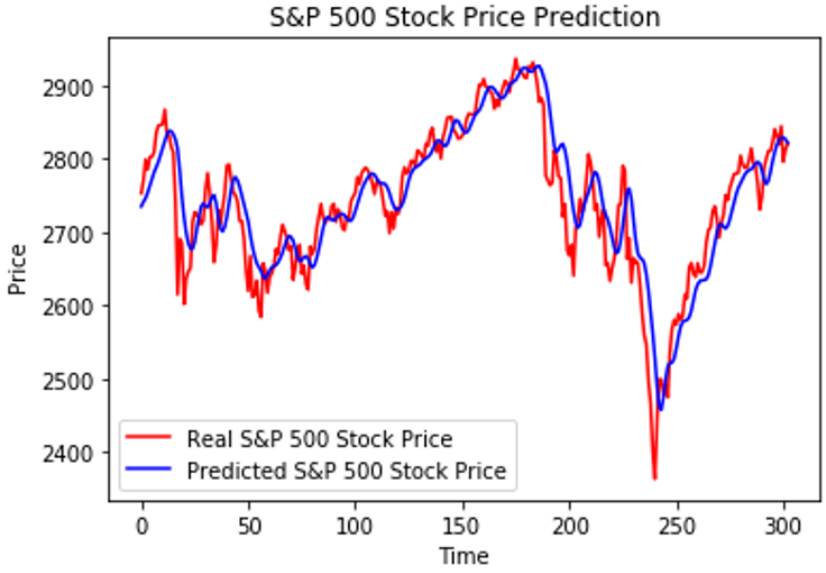

Figure 7.4: Graph of real and predicted price of SP 500 index using multiple layer perceptron (MLP)

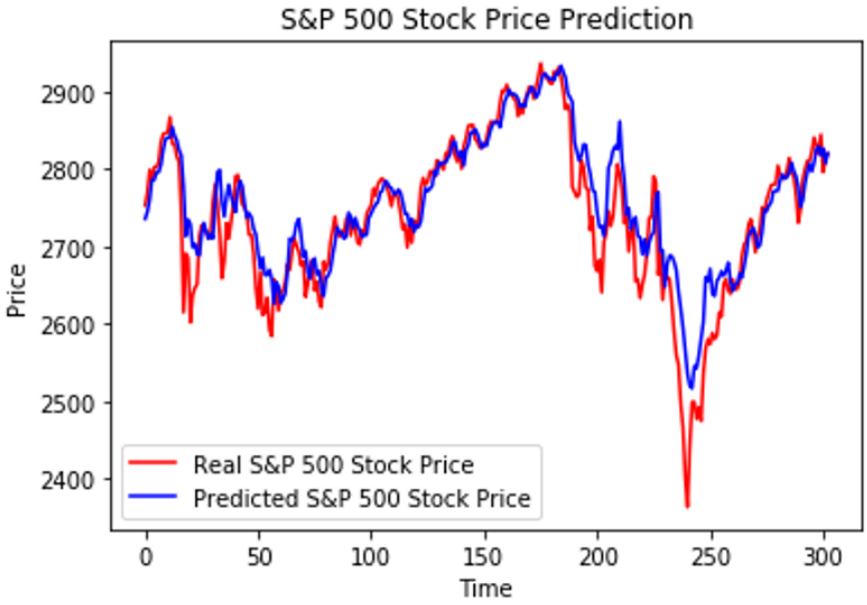

Figure 7.5: Graph of real and predicted price of SP 500 index using long short term memory (LSTM)

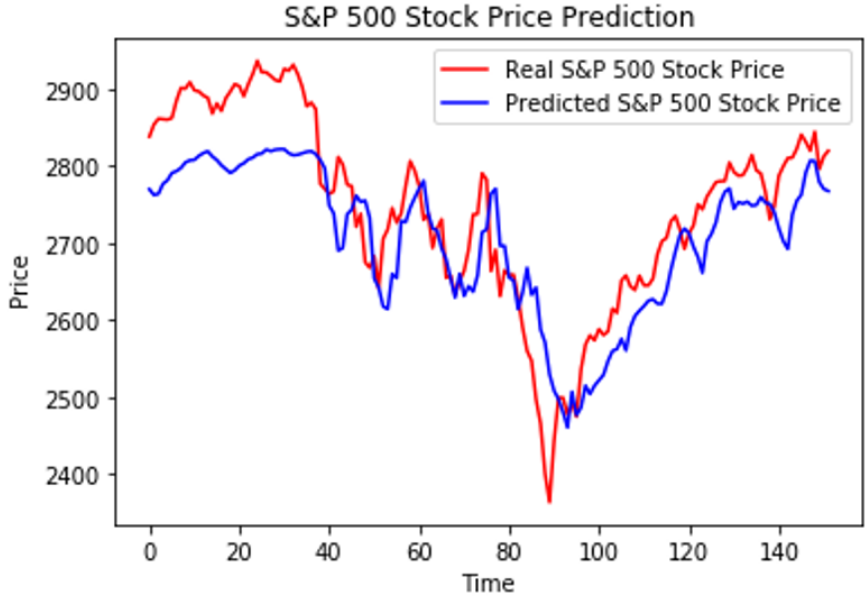

Figure 7.6: Graph of real and predicted price of SP 500 index using long convolutional neural network (CNN)

Chapter 8

Conclusion

Over the years, neural network has emerged as one of the most important field of study due to its flexible nature to adapt with different types of data and make a better result out of it. This paper attempts to analyze and study the intuition behind neural network. We discussed about the structure and why it is useful in huge amount of nonlinear data. We described in depth about how neural network does the computation of input data in the hidden layers and find the optimum result. Later we consider the fact that we can improve the performance of neural network and make a better model by tweaking the value of different types of hyper parameters like learning rate, activation function, number of layer, node or epoch etc. We can also use bias and variance, moving average, regularization, normalization to make faster and more accurate model which we discussed in chapter 3. In the end we implemented various architecture of neural network on financial data. We used nonlinear financial data series to forecast stock price with different artificial neural network model. In this study we experimented three different artificial neural network architecture on two different stock market dataset which are Amazon stock price and SP 500 indexes. The artificial neural network models we used, are multiple layer perceptron, long short term memory network and convolutional neural network. By using these three model, we trained our network and tested on test data. Then we calculated mean squared error, mean absolute error and $R^2$to compare the how good the prediction is for different model. Further we showed graphical view of predicted data and actual data of three model on both dataset. From the result we had, we found that though CNN can find pattern in image but it is not quite good when the data is sequential. MLP and LSTM have given better prediction result than CNN. Additionally we observed LSTM is actually more useful to make sense of sequential data. From the study it is quite clear that neural network has great capability to capture the relation of features in a data and can be used more flawlessly in future. However, it is not perfect today but it can be possible one day that neural network model have perfect understanding on noisy and chaotic data of financial time series.

Bibliography

1. Fang, H; Lai, S. and Lai, M. (1994), "Fractal structure in currency futures price dynamics". Journal of Futures Markets, 14, pp. 169-181.

2. Bollerslev, T. R. (1986), "Generalized Autoregressive Conditional Heteroskeda sticity". Journal of Econometrics, 51, pp. 307-327.

3. Lawrence, J. (1991), "Introduction to Neural Networks". California Scientific Sortware: Grass Valley, CA.

4. Minsky, M. and Papert, S. (1969), "Perceptrons". MIT Press, Cambridge, MA.

5. Nelson, M. M. and Illingworth. (1991), "A Practical Guide to Neural Nets". Addison Wesley, Reading, MA.

6. Wong, B. K. and Selvi, Y. (1998), "Neural network applications in business: A review and analysis of the literature". Information & Management, 34, pp. 129-139.

7. Chatterjee, A.; Ayadi, O. F. and Boone, B. E. (2000), "Artificial neural network and the financial markets: A survey". Managerial Finance, 26, pp. 32-45.

8. G. Zhang and M. Y. Hu, Neural network forecasting of the British pound/US dollar exchange rate, Journal of Management Science 26 (1998) 495–506.

9. MumtazIpek, UfukBolukbas, and EminGundogar, (2011), Estimating Financial Failure of the TurkishBanks Using Artificial Neural Networks, the 2011 New Orleans International Academic Conference New Orleans, Louisiana USA 629.

10. A. Fadlalla and C. H. Lin, An analysis of the applications of neural networks in Finance, Interfaces 31(4) (2001) 112–122.

11. Anyaeche C. O. and Ighravwe D. E., (2013), Predicting performance measures using linear regression and neural network: A comparison, African Journal of Engineering Research Vol. 1(3) , pp. 84-89,

12. T. Liu, S. Fang, Y. Zhao, P. Wang and J Zhang," Implementation of Training Convolutional Neural Networks"

13. Jones,E. (2004), "An Introduction to Neural Network". United States of America.

14. Walia, A. (2019). “Activation functions and it’s types-Which is better?” Available at: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f [Accessed 6 Apr. 2019].

15. Chávez, G. (2018). “Understanding Logistic Regression”.

16. Swaminathan, S. (2018). Logistic Regression — Detailed Overview.

17. Bottou, L. (2019). Large-Scale Machine Learning with Stochastic Gradient Descent. NEC Labs America, Princeton NJ 08542, USA

18. SHARMA, S. (2017). “Activation Functions in Neural Networks”.

19. Hao, Z. (2017). “Activation Functions in Neural Networks”, Singapore.

20. Bullinaria, J. (2004). Bias and Variance, Under-Fitting and Over-Fitting in Neural Network.

21. Jain, S. (2018). “An Overview of Regularization Techniques in Deep Learning”.

22. Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv preprint arXiv: 1502.03167v3, 2015.

23. Ruder, S. (2011). An overview of gradient descent optimization algorithms. Dublin.

24. Ning Qian. On the momentum term in gradient descent learning algorithms. Neural networks :the official journal of the International Neural Network Society,12 (1):145–151, 1999.

25. Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, pages 1–13, 2015

26. Banerjee, S. (2018, May 23). An Introduction to Recurrent Neural Networks. Retrieved from-https://medium.com/explore-artificial-intelligence/an-intro duction-to-recurrent-neural-networks-72c97bf0912

27. Olah, C. (2015, August 27). Understanding LSTM Networks. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

28. Hochreiter, S. and Schmidhuber, J. (1997),Long Short-term Memory, Neural computation 9(8):1735-1997

29. H. Sak, A. Senior and F. Beaufays, ”Long Short-term Memory Recurrent Neural Network Architecture for Large Scale Acoustic Modeling”,USA

30. Saha, S. (2018, December 15). A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53