**Multidimensional Neural Networks**

Project Work II Report

Submitted in partial fulfilment of requirement for the Degree of

**BACHELOR OF TECHNOLOGY**

IN

**COMPUTER SCIENCE & ENGINEERING**

BY

EN18CS301289 Venish Patidar

EN19CS3T1004 Raj Gupta

EN18CS301294 Yash Biyani

**Under the Guidance of**

Prof. Preetesh Purohit



Department of Computer Science & Engineering

Faculty of Engineering

MEDI-CAPS UNIVERSITY, INDORE- 453331

May 2022

Copyright ©

This work is the intellectual property of the author Venish Patidar. Unless otherwise stated you can use 10% of this work for private study, personal or non-commercial research. Design, idea, approach, neural model and implementation are owned and controlled by author. Any reuse of information unless otherwise stated should be fully referenced, quoting the author, title, university, degree level and pagination. For more information, queries and request to get a substantial copy of project contact owner.

Report Approval

The project work **“MULTIDIMENSIONAL NEURAL NETWORKS”** is hereby approved as a creditable study of a computer science and engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite for the Degree for which it has been submitted.

It is to be understood that by this approval the undersigned do not endorse or approved any statement made, opinion expressed, or conclusion drawn there in; but approve the “Project Report” only for the purpose for which it has been submitted.

**Internal Examiner**

Name:

Designation:

Affiliation:

**External Examiner**

Name:

Designation:

Affiliation:

Declaration

We hereby declare that the project entitled **“MULTIDIMENSIONAL NEURAL NETWORKS”** submittedin partial fulfilment for the award of the degree of Bachelor of Technology “Computer Science and Engineering” completed under the supervision of **Prof. Preetesh Purohit,** Faculty of Engineering, Medi-Caps University Indore is an authentic work.

Further, we declare that the content of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma.

**Signature and name of the student(s) with date**

**Venish Patidar**

**Raj Gupta**

**Yash Biyani**

Certificate

I, **Prof. Preetesh Purohit** certify that the project entitled **“MULTIDIMENSIONAL NEURAL NETWORKS”** submittedin partial fulfilment for the award of the degree of Bachelor of Technology of Computer Science and Engineering by **Venish Patidar, Raj Gupta and Yash Biyani** istherecordcarried out by them under my guidance and that the work has not formed the basis of award of any other degree elsewhere.

**Internal Examiner**

Name:

Computer Science & Engineering

Medi-Caps University, Indore

**External Examiner**

Name:

Designation:

Affiliation:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Dr. Pramod S. Nair**

Head of the Department

Computer Science & Engineering

Medi-Caps University, Indore

Acknowledgements

First of all, we would like to thank God for giving me the strength, intelligence, ability and inspiration to complete this project. Without him we would not have been able to do this project.

There are many other people that helped me during this project as well to whom we must endure and express my gratitude for their emotional and economical contributions to the successful completion of this dissertation.

We would like to express my deepest gratitude to Honourable Chancellor, Shri R C Mittal, who has provided me with every facility to successfully carry out this project, and my profound indebtedness to Prof. (Dr.) Dilip K. Patnaik, Vice Chancellor, Medi-Caps University, whose unfailing support and enthusiasm has always boosted up my morale. We also thank Prof. (Dr.) D K Panda, Pro Vice Chancellor, Dr. Suresh Jain, Dean Faculty of Engineering, Medi-Caps University, for giving me a chance to work on this project. We would also like to thank my Head of the Department Dr. Pramod S. Nair for his continuous encouragement for betterment of the project.

We would like to thank and my sincere appreciation to my guide Prof. Preetesh Purohit, who greatly supported me for completing this project. We will be grateful to his guidance and assistance for life. My heartiest thank to family, for encouraging and supporting me throughout the project. We would like to thank my batchmates and all friends for encouraging and supporting me.

Thank you all without your support and guidance this project would not have been possible.

Venish Patidar,

Raj Gupta,

Yash Biyani,

B.Tech. CSE-E IV Year,

Department of Computer Science & Engineering

Medi-Caps University, Indore

Abstract

The area of neural network field is astronomical and it is by the time today broadening its horizons. As due the fact of its vast research area, there are lot of different types of neural network architecture. All of which varying from having a single neuron, which is perceptron, to all the way having multiple neuron and hidden layers. Some are feedforward neural networks while other are feedback compatible as well. However, all of this neural network architecture involves a single computational node called neuron, which is core computational unit of any architecture. Each neuron has an activation function which act as switch i.e., weather to fire the neuron or not and carry on the information to next layer. Furthermore, almost all neural network architectures comprise of different hidden layers are made up of it and having same activation function over any particular hidden layer. Hence, inhibiting from using multiple activation function on same layer for different sets of neurons. Therefore, introduction to a new dense layer on same level but on different dimension will allow us to use multiple activation functions on same level, while receiving inputs from same previous level and calculating information with multiple activation function before passing it to next level in architecture thus called multidimensional neural network, which will also be compatible with conventional dense layer as well. Finally, resulting in attaining higher accuracies in less amount of training and having quicker loss convergence.

Table of Contents

[Copyright © ii](#_Toc100619740)

[Report Approval iii](#_Toc100619741)

[Declaration iv](#_Toc100619742)

[Certificate v](#_Toc100619743)

[Acknowledgements vi](#_Toc100619744)

[Abstract vii](#_Toc100619745)

[Table of Contents viii](#_Toc100619746)

[List of Figures x](#_Toc100619747)

[Abbreviations xi](#_Toc100619748)

[Chapter 1 1](#_Toc100619749)

[Introduction 1](#_Toc100619750)

[1.1 Research background and motivation 1](#_Toc100619751)

[1.2 Statement of problem 2](#_Toc100619752)

[Chapter 2 3](#_Toc100619753)

[Multidimensional Neural Network 3](#_Toc100619754)

[2.1 Multidimensional layer 4](#_Toc100619755)

[2.2 Weights and biases 5](#_Toc100619756)

[2.3 Mathematics behind MDNN 6](#_Toc100619757)

[2.3.1 Feedforward 6](#_Toc100619758)

[2.3.2 Backpropagation 7](#_Toc100619759)

[Chapter 3 10](#_Toc100619760)

[Litreature review 10](#_Toc100619761)

[3.1 Matrix multiplication 10](#_Toc100619762)

[3.2 Use of Operator 11](#_Toc100619763)

[3.3 Activation Functions 11](#_Toc100619764)

[3.3.1 Linear 12](#_Toc100619765)

[3.3.2 Sigmoid / Logistic 12](#_Toc100619766)

[3.3.3 Tanh 13](#_Toc100619767)

[3.3.4 ReLU 13](#_Toc100619768)

[3.4 Loss Functions or Cost Functions 14](#_Toc100619769)

[3.5 Optimizers 14](#_Toc100619770)

[3.6 Partial derivatives 14](#_Toc100619771)

[Chapter 4 15](#_Toc100619772)

[MDNN Desgin, implementation And Documentation 15](#_Toc100619773)

[4.1 MDNN’s nested classes 15](#_Toc100619774)

[4.2 MDNN’s member functions 16](#_Toc100619775)

[4.3 MDNN’s member variable 17](#_Toc100619776)

[4.4 Documentation 17](#_Toc100619777)

[4.4.1 MDNN Classes 18](#_Toc100619778)

[4.4.2 MDNN member functions 19](#_Toc100619779)

[Chapter 5 23](#_Toc100619780)

[Experiments and Results 23](#_Toc100619781)

[Chapter 6 26](#_Toc100619782)

[Conclusion And Future Work 26](#_Toc100619783)

[Appendix 27](#_Toc100619784)

[A.1 Example program for Planner data classification 27](#_Toc100619785)

[A.2 Statements and Declarations 28](#_Toc100619786)

[A.2.1 Funding 28](#_Toc100619787)

[A.2.2 Conflict of interest 28](#_Toc100619788)

[A.2.3 Ethics approval 28](#_Toc100619789)

[A.2.4 Consent to participate 28](#_Toc100619790)

[A.2.5 Consent for publication 28](#_Toc100619791)

[A.2.6 Code availability 28](#_Toc100619792)

[A.2.7 Authors’ contributions 28](#_Toc100619793)

[Bibliography 30](#_Toc100619794)

List of Figures

[Fig 1.1 Perceptron 1](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533796)

[Fig 1.2 Conventional Multilayer Neural Network 2](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533797)

[Fig 2.1 Multidimensional Neural Network 3](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533798)

[Fig 2.2 Multidimensional layer 4](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533799)

[Fig 2.3 Weights mapping 5](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533800)

[Fig 2.4 CALCULATING gradients of weights 7](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533801)

[Fig 3.1 Matrix multiplication 10](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533802)

[Fig 3.2 Linear Activation Function 12](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533803)

[Fig 3.2 Sigmoid / Logistic Activation Function 12](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533804)

[Fig 3.2 Tanh Activation Function 13](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533805)

[Fig 3.2 Relu Activation Function 13](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533806)

[Fig 4.1 Class hierarchy of MDNN 16](file:///D:\Project\Multidimensional%20Neural%20Networks.docx#_Toc100533807)

[Fig 5.1. (**A**) Planner dataset consisting 200 red dots and 200 blue dots. (**B**) Losses plot for both mdnn and conv on different weight sets all combines into one. (**C**) Accuracies plot for both mdnn and conv on different weight sets all combines into one. 23](#_Toc100533808)

[Fig 5.2. Losses, accuracies, Time taken per epoch and Decision boundary plot for MDNN and CONV neural network 24](#_Toc100533809)

Abbreviations

MDNN Multidimensional Neural Network

MDL Multidimensional layer

DL Dense layer

CONV Conventional Neural Network

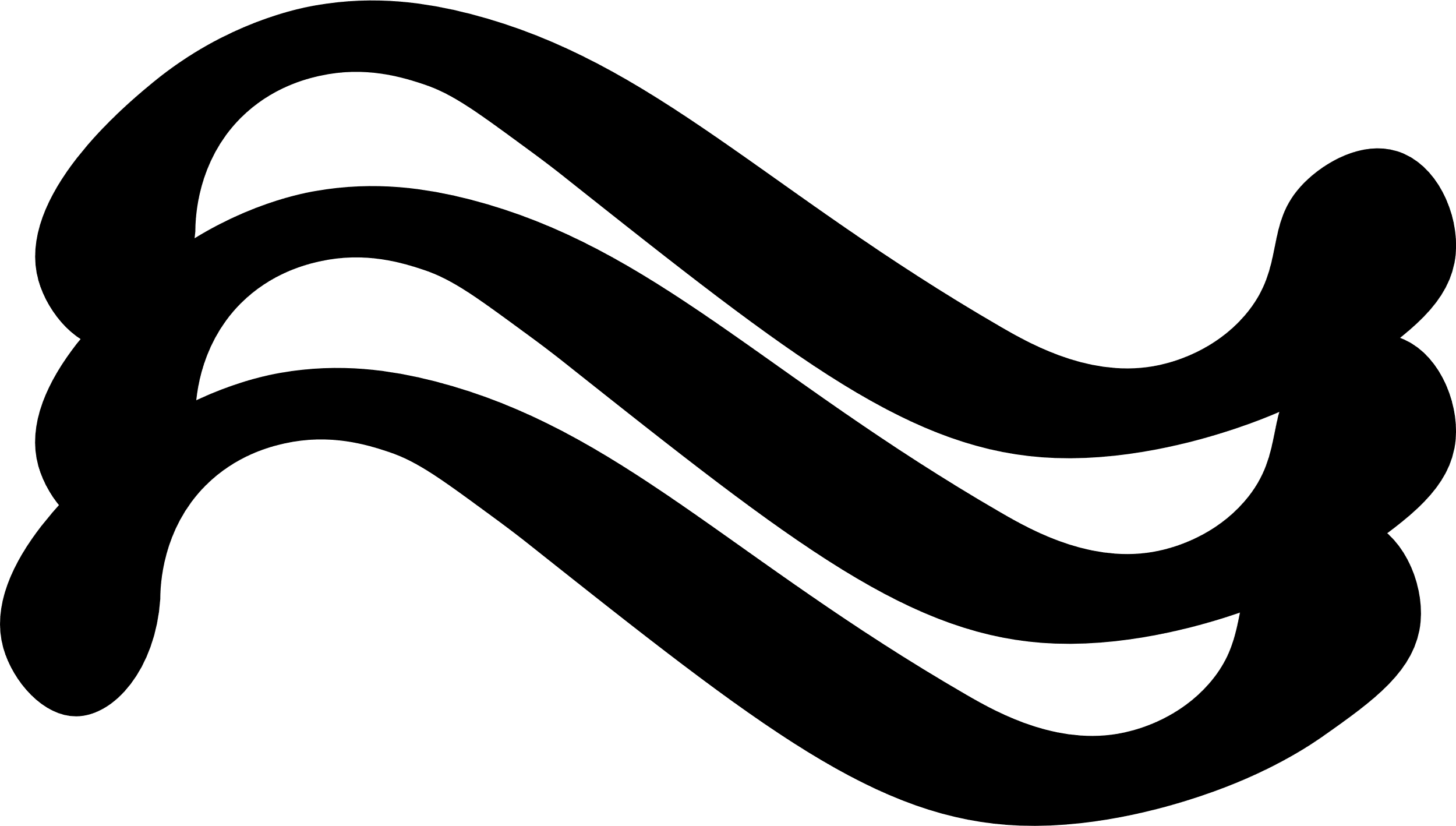
TTPE Time taken per epoch

BCE Binary Cross Entropy

SGD Stochastic Gradient Descent

Chapter 1

**Introduction**



T

he end points of neural network field is still unknown and it’s by the time today expanding. As due the fact of its vast research area, there are lots of different type of neural network architecture. All varying from a single neuron which is perceptron, all the way to RNNs (recurrent neural networks). Some are feedforward neural nets others are feedback compatible as well. However, most of these neural network architectures have multiple hidden layers, comprises in itself, there isn’t any dimensionality feature to these hidden layers, i.e., multiple layers stacked-on-each-other (metaphorically) to make a single hidden layer, as to be called Multidimensional neural network. The Multidimensional neural network is research project work which aims to explore the area of neural network architecture. The proposed architecture will make the existing network more flexible and introduce new dimensionality parameter to it, particularly to hidden layers. It does so by introducing a whole complete new hidden layer in another dimension to a particular layer. Introduction to the new hidden layer in another dimension increases parameters and complexity, however, as it stands now, the accuracy and loss will be explored further.

1.1 Research background and motivation

W1

W2

WN

X1

X2

XN

Y

Fig 1.1 Perceptron

The first artificial neuron was the Threshold Logic Unit (TLU), proposed by Warren McCulloch and Walter Pitts in 1943 also known by its infamous term perceptron. This perceptron consists of a summing function and an activation function [1]. Soon after discovery of the perceptron, multilayer neural networks were already being implemented. These neural networks consist of stack perceptron at different hidden layer and at same time connected to each other with trainable weights and biases [2]. Underlying this basic perceptron there are lots of different neural network architectures. Such as CNNs, RNNs and many other [3]. There are astonishing results that have been achieved by these architectures, be it object detection [4] or language translation [5], this ANNs showed that they are capable of achieving more.

Input

Hidden layer 1

Hidden layer 2

Hidden layer 3

Output

Fig 1.2 Conventional Multilayer Neural Network

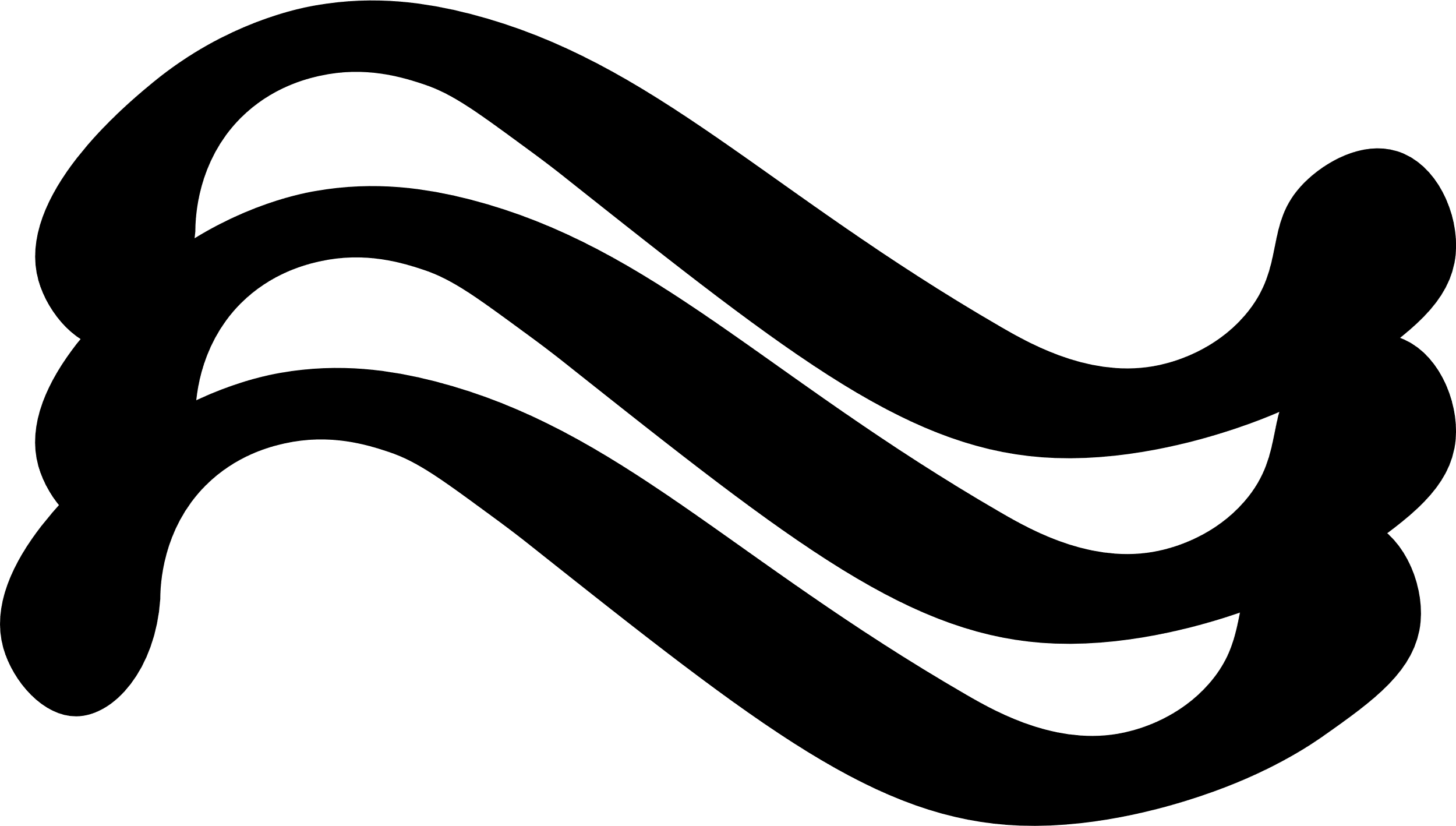
In spite of all this, there is no change in multilayer fully connected neural network to this date, which holds back us from using multiple activation function on any particular level in architecture. The Multidimensional neural network aims to make multi-activation hidden layer feasible. The proposed architecture will make the existing network more flexible by introducing a new dimensionality parameter to it, more specifically to hidden layers. It does so by adding a complete whole new hidden layer in another dimension to that particular layer. Introduction to the new hidden layer in another dimension results in facilitating using of multi-activation function on same level, while, each dimension on any particular level receiving same input. The underlying math will be discussed in further sections.

***1.2 Statement of problem***

Design and implement a new neural network architecture, that is capable of having more than one dimensions in its hidden layers as that of conventional neural network. One challenge would be updating weights of hidden layer appropriately, that will have been adjusted by loss function. Another challenge would be to properly implement a loss function with various loss function such as RMS and the others however, while implementing this loss function this architecture should also be flexible towards learning rates and implement learning algorithms appropriately. Other challenge would be making this multidimensional hidden layer to able to connect with conventional hidden layer as well, ultimately the other challenge would be to increase the accuracy as to of conventional architecture.

Chapter 2

**Multidimensional Neural Network**



Input

Dense layer

Multidimensional layer

Multidimensional layer

Multidimensional layer

\3\

Dense layer

output

Fig 2.1 Multidimensional Neural Network

W

e propose a multidimensional neural architecture by adding new dimension in hidden layer of conventional neural networks as shown in figure 2.1. In our proposal any layer with having more than one dimension would be termed as multidimensional layer MDL and any layer having one dimension would be termed as one dimensional or dense layer DL. The multidimensional layer would consist of multiple one-dimensional layer or dense layer in another dimension. Each layer would be termed MDL or DL followed by layer number see figure 2.1. Every layer be it MDL or DL will be densely connected, (i.e., each neuron of previous layer is connected to each neuron of next layer [6]), to each other in following four possible ways.

1. **DL – DL**

A dense layer followed by dense layer; they will be get connected usually i.e., densely connected. See figure 2.1 DL\_5 and DL\_6.

1. **MDL – DL**

Multidimensional layer followed by dense layer; dense layer will be densely mapped to all the dimensions on previous multidimensional layer. See figure 2.1 MDL\_1 and DL\_2.

1. **DL – MDL**

Dense layer followed by multidimensional layer; each dense layer in MDL will densely mapped and receive input from same previous dense layer. See figure 2.1 DL\_2 and MDL\_3.

1. **MDL – MDL**

Multidimensional layer followed by multidimensional layer; each dimension in both MDLs will be connected to each other. See figure 2.1 MDL\_3 and MDL\_4.

Every MDL will have same number of neurons in its each dimension in dense network however, there will be different matrices of weights and biases for each dense layer to other dense layer.

***2.1*** Multidimensional layer

Each multidimensional layer comprises of dense layer [7] side by side as illustrated in figure 2.2. Each of these dense layers consists of neurons. The neuron gets input from previous layer and then after performing computation it further computes its activation thorough activation function and then it produces output. Every dense layer in multi-dimensional in our proposal will be a matrix, where each element of matrix will be a corresponding value of that position’s neuron, which will be referred by:

Fig 2.2 Multidimensional layer

Neuron

Dense layer at 1st dimension

Dense layer at 2nd dimension

MDL\_3

*X ((layer number) (dimension number-1))*

The multidimensional layer from figure 2.2 is the 3rd layer, indexing starts from zero, of the neural network from figure 2.1. The dense layer at 1st dimension will be referred as *X30* and the dense layer at 2nd dimension will be referred as *X31.* Similarly, activation function of the corresponding layer at any dimension would be referred by:

Z ((layer number) (dimension number-1))

The activation function at dimension one in figure 2.2 would be referred as Z30 and at dimension two would be referred as Z31.

***2.2*** Weights and biases

Weights of any particular layer will be referred as:

W ((layer number) (to dimension) (from dimension))

And Biases:

B ((layer number) (to dimension) (from dimension))

In figure 2.1 Mapping of weights of layer MDL\_1 at dimension one to input layer I\_0 will be referred as W100 and at dimension two of MDL\_1 to input layer I\_0 W101, similarly biases would be referred as B100 and B101 respectively. The dimension of any weight matrix at any particular layer will be:

(Current layer neuron, previous layer neuron)

X20

X30

DL\_2

DL\_3

W300

Fig 2.3 Weights mapping

Each element of weight matrix represents a connection from previous layer to current, which will be referred by:

*W [to layer neuron, from layer neuron]*

For figure 2.3 the weight matrix for layer DL\_3 will be defined as:

Similarly, dimension of biases will be (Current layer neuron1), for fig 2.3 biases would be:

***2.3*** Mathematics behind MDNN

2.3.1 Feedforward

For the mapping of DL to DL the feedforward calculation follows conventional methods [8] but wherever MDL is introduced the calculation gets slightly changed; following are the three possible ways:

1. **DL – MDL**

Each dense layer in MDL will receive same input from previous dense layer but weights and biases are different for each dense layer in MDL.

1. **MDL – DL**

The dense layer DL will receive input from all the dimensions of previous MDL, each having different activation function and then it will perform summation over it see equation 1.

1. **MDL – MDL**

Each dense layer in MDL will get input from all of the dense layers from the previous layer see equation 2.

Mathematically, feedforward calculations for any particular layer would be defined as:

Where, CLN = Current layer number

CDN = Current dimension number

PLD = Previous layer dimensions

PLN = Previous layer number

Here W, X and B are matrix of weights, layer values and biases respectively, while Z is activation function of that layer. In figure 2.1 calculation for layer DL\_2 according to above mentioned formula would be:

X20 = Z20{[(W200X10) +B200] + [(W210X11) +B210]} (1)

Similarly, for layer MDL\_4 and dimension 2:

X41 = Z41{[(W401X30) +B401] + [(W411X31) +B411]} (2)

***2.3.2*** Backpropagation

In backpropagation loss calculation follows conventional way [9] by using loss function *J* [10] where, *y* is predicted or generatedoutput from network and is expected output. However, weights gradient and biases gradient [11] are redefined as due to introduction of new multidimensional layer to the conventional neural networks.

DL\_5

DL\_4

MDL\_3

Fig 2.4 CALCULATING gradients of weights

*.*

*.*

*.*

Let us consider a piece of any MDNN figure 3.1 and calculate gradient of the given weights which is between MDL\_3 dimension one to DL\_4 i.e., . Losses for figure 3.1 will be calculated by loss function *J* where, *y* is X50 and is expected output. Calculating partial derivative of loss function J with respect to weight matrix :

Where is,

and

Let us say,

Here

also let us say,

Here denotes element wise multiplication.

Generalizing it for any weighted matrix:

Similarly, it can be derived for any bias matrix B:

Where, for each layer except output layer is:

While for output layer it is calculated by:

Where, CLN = Current layer number

CDN = Current dimension number

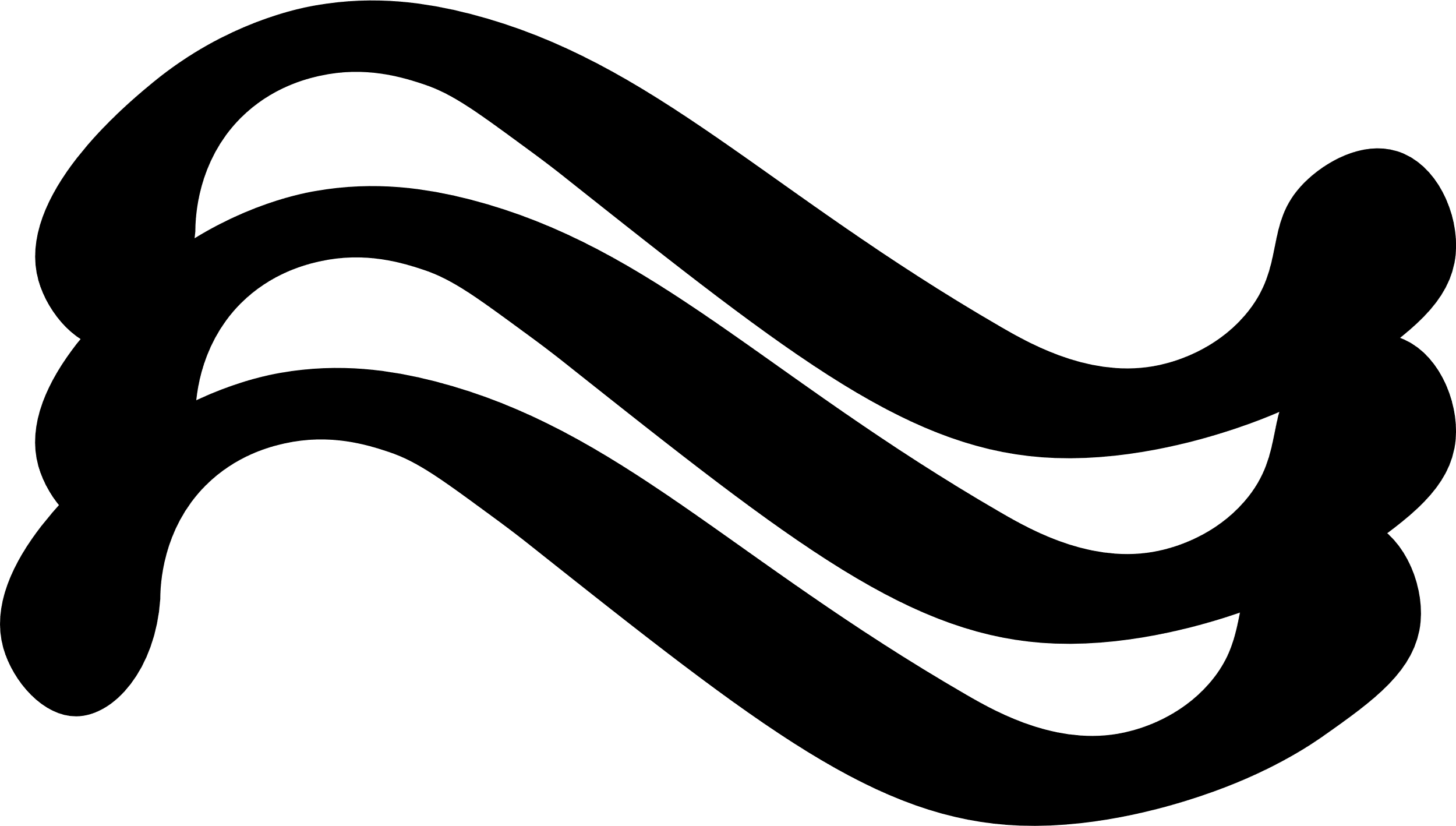
PLDN = Previous layer dimension number

PLN = Previous layer number

NLD= Next layer dimensions

Chapter 3

**Litreature review**



T

he base of MDNN architecture is the art of matrix calculations. Most of the data calculated in MDNN is through matrices. The matrix is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns, which is used to represent a mathematical object or a property of such an object. Every dense layer in MDNN is one dimensional matrix having length of number of neurons it comprises. On the other hand, each multidimensional layer is two-dimensional matrix each column refers to dimension and each row as neurons. Similarly, weight matrices are two dimensional and bias matrices are one dimensional.

In spite of matrices the crucial role in each layer is of activation functions. Moreover, MDNN is all about enabling the user to use multi-activated layer in their neural network. We will discuss more about activation functions in further sections along with another important function that is, loss function sometime also called as cost function.

***3.1*** Matrix multiplication

In mathematics, particularly in linear algebra, matrix multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, known as the matrix product, has the number of rows of the first and the number of columns of the second matrix. The product of matrices A and B is denoted as AB. -Wiki.

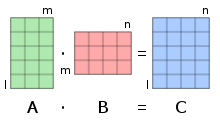
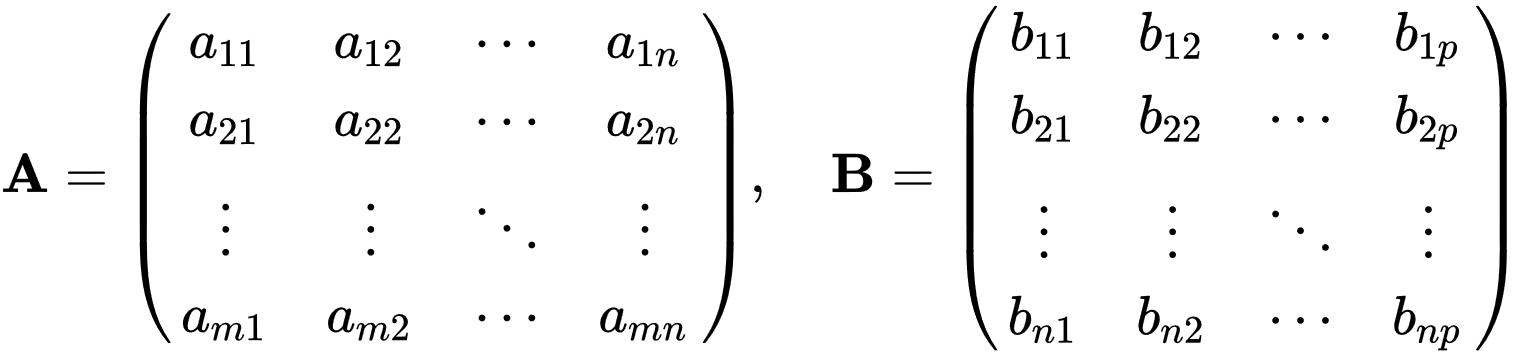
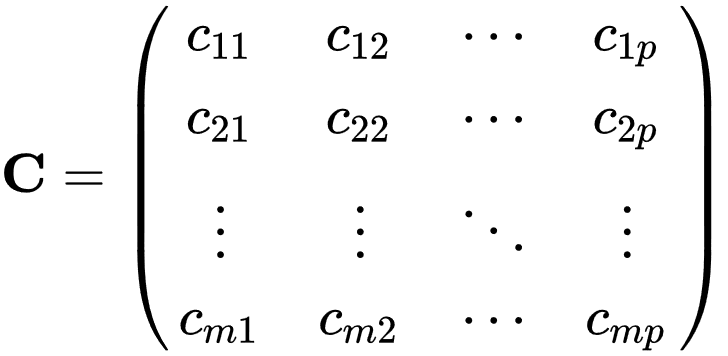


Fig 3.1 Matrix multiplication

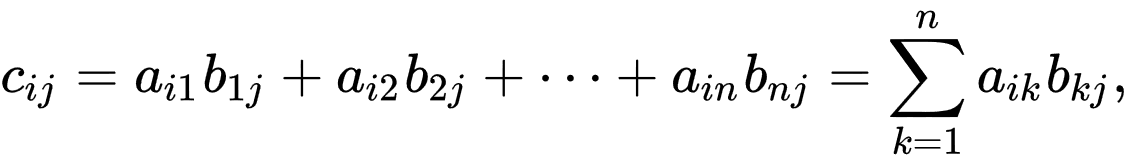
If A is an m × n matrix and B is an n × p matrix,



the matrix product C = AB (denoted without multiplication signs or dots) is defined to be the m × p matrix



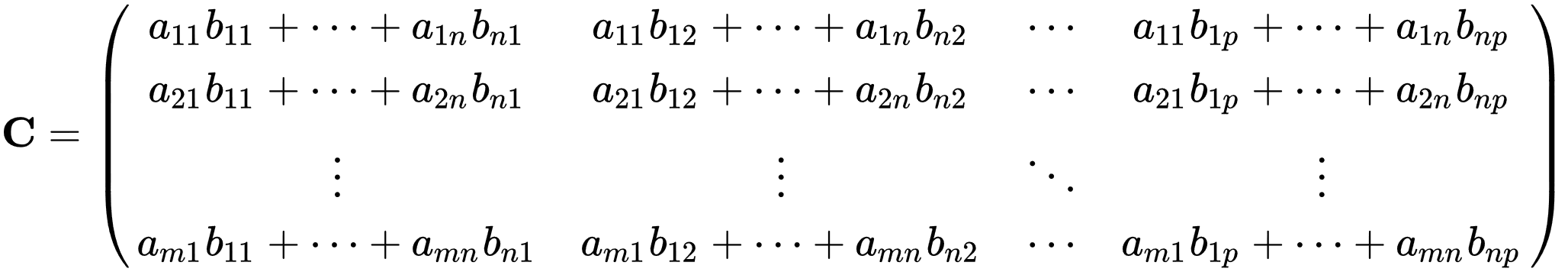
such that,



for i = 1, ..., m and j = 1, ..., p.

That is, the entry Cij of the product is obtained by multiplying term-by-term the entries of the ith row of A and the jth column of B, and summing these n products. In other words, Cij is the dot product of the ith row of A and the jth column of B.

Therefore, AB can also be written as



Thus, the product AB is defined if and only if the number of columns in A equals the number of rows in B.

***3.2 Use of***  Operator

denotes element-wise matrix multiplication (also known as Hadamard Product), which is every element of the first matrix is multiplied by the second matrix's corresponding element. When performing the element-wise matrix multiplication, both matrices should be of the same dimensions

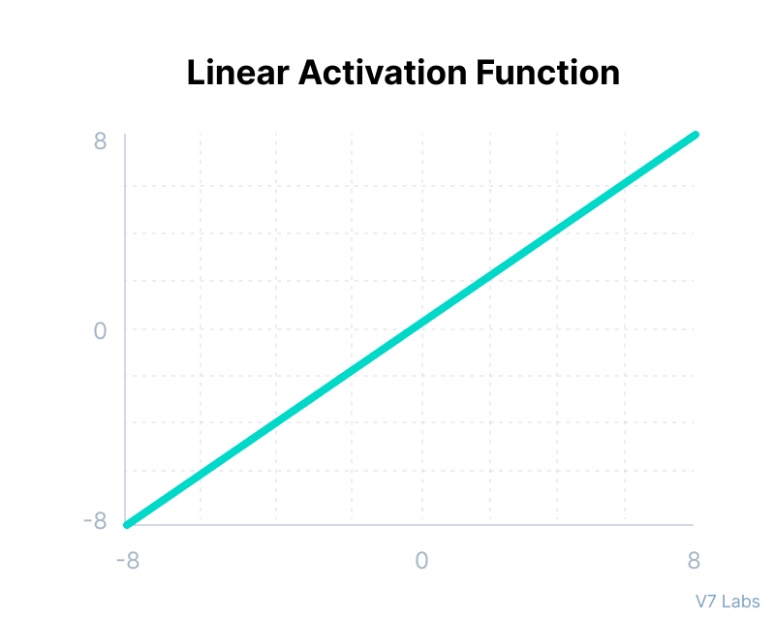
***3.3 Activation Functions***

Activation functions are mathematical equations that determine the output of a neural network, The most common activation functions and those we will be using in MDNN are four commonly used function which are described as follow.

***3.3.1 Linear***

The linear activation function is also known as Identity Function where the activation is proportional to the input.

Fig 3.2 Linear Activation Function



Mathematically it can be represented as:

***3.3.2 Sigmoid / Logistic***

This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

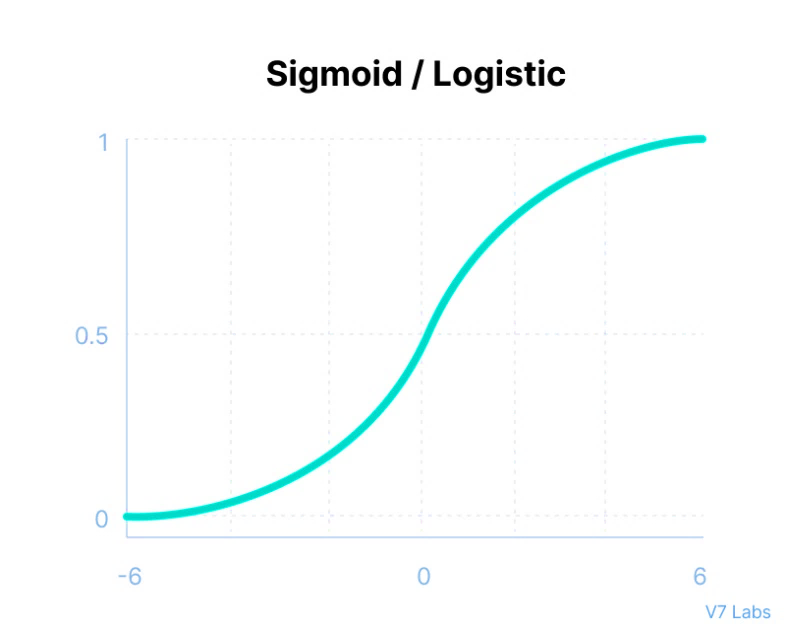


Fig 3.2 Sigmoid / Logistic Activation Function

Mathematically it can be represented as:

***3.3.3 Tanh***

Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

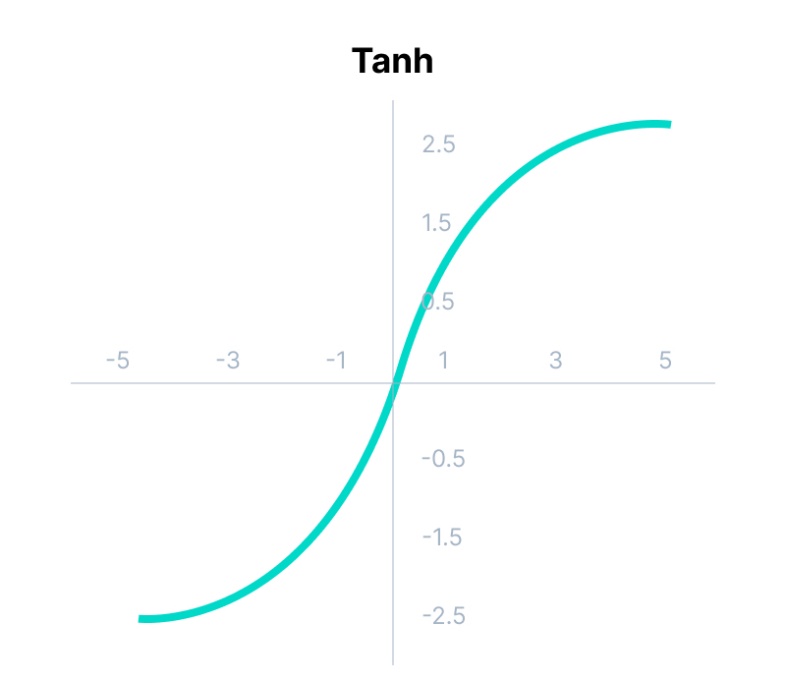


Fig 3.2 Tanh Activation Function

Mathematically it can be represented as:

***3.3.4 ReLU***

Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient. The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.

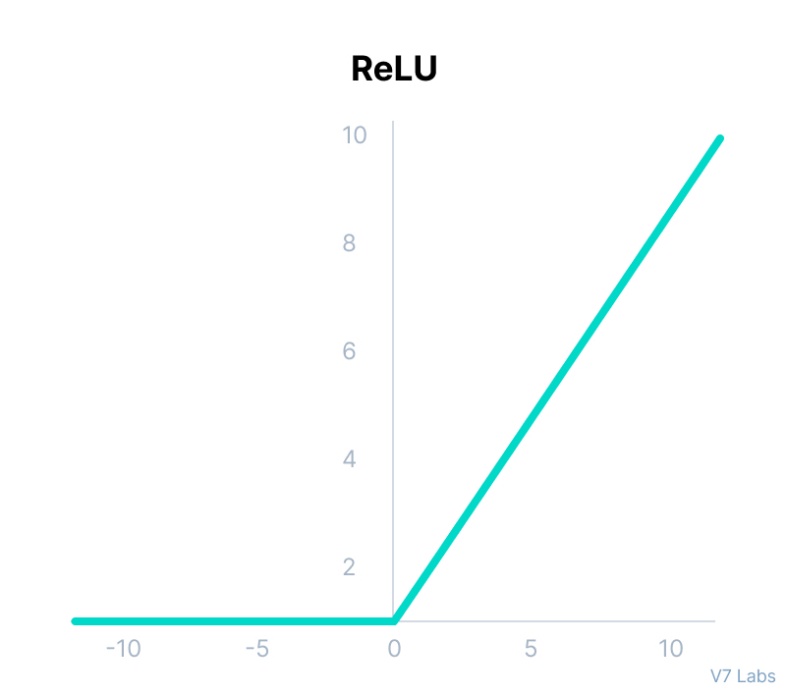


Fig 3.2 Relu Activation Function

Mathematically it can be represented as:

***3.4 Loss Functions or Cost Functions***

In mathematical optimization and decision theory, a loss function or cost function (sometimes also called an error function) is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event [9 10]. Some of cost functions are Binary cross entropy, categorical cross entropy, SoftMax and more. Mostly we will be using BCE in MDNN. Binary cross entropy mathematically is defined as:

***3.5 Optimizers***

While training the deep learning model, we need to modify each epoch’s weights and minimize the loss function. An optimizer is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy [12]. The most common optimizers are Gradient Descent, Stochastic Gradient Descent, Adam, RMSProp and others. For our testing purposes we had used Stochastic Gradient Descent which mathematically defined as:

Where is learning rate and are the weight gradients that been calculated through backpropagation as talked earlier.

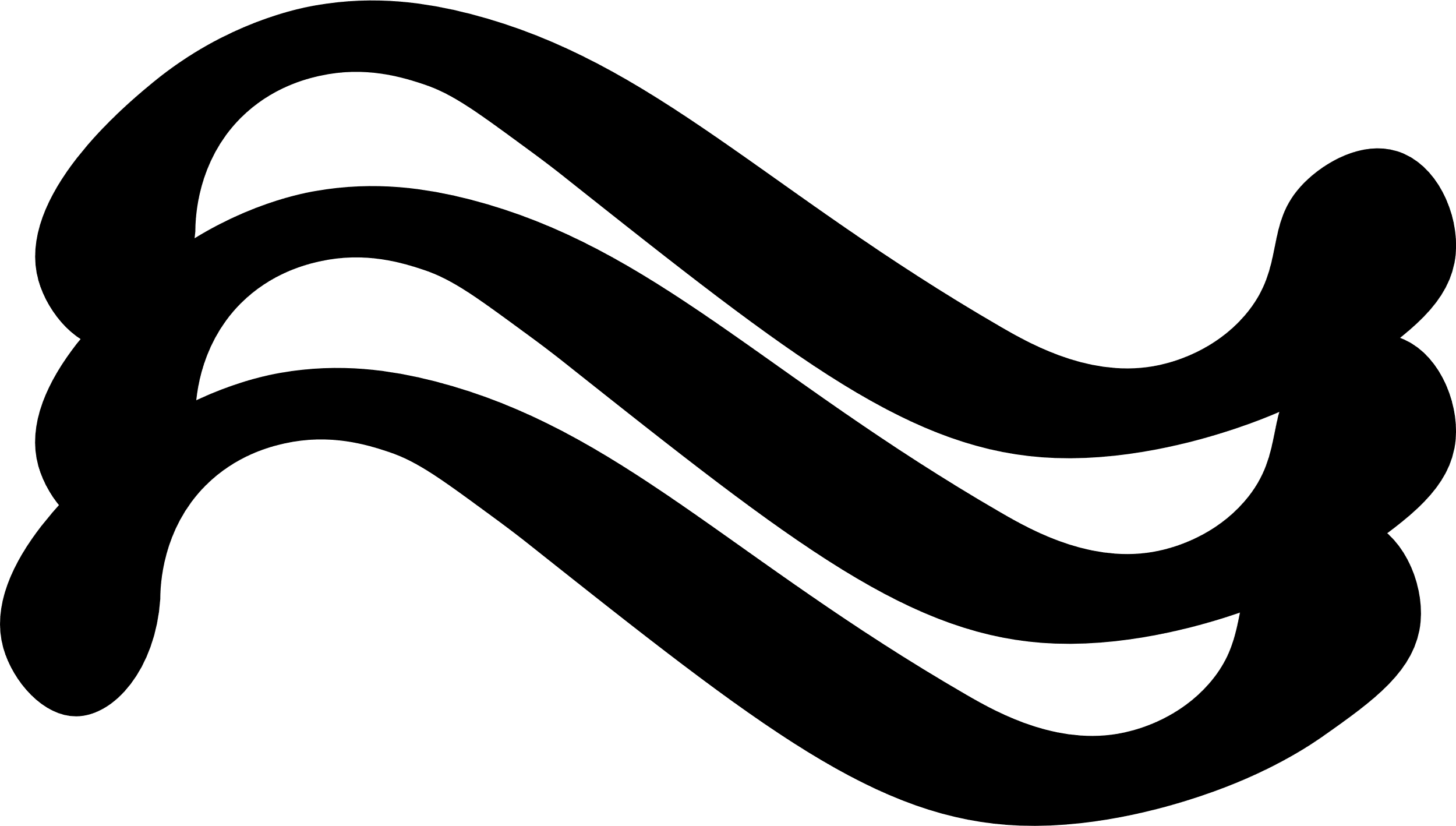
***3.6 Partial derivatives***

In mathematics, a partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant (as opposed to the total derivative, in which all variables are allowed to vary). Partial derivatives are used in vector calculus and differential geometry.

The partial derivative of a function with respect to the variable is variously denoted by:

Chapter 4

**MDNN Desgin, implementation And Documentation**



M

DNN designed in such a way that it is compatible with conventional neural networks as well. It has classic class-object design which is simple yet very effective and robust. The MDNN is implemented in C++ language with Armadillo the only required library. Armadillo is a high-quality linear algebra library (matrix maths) for the C++ language, aiming towards a good balance between speed and ease of use. The whole program is compiled and packaged using GNU compiler version 11.2.0. The source program consists of the class named MDNN. This MDNN class is responsible for creating an object, this object is then used to create the different neural models see appendix A.1. Let’s explore the heart, the brain and the motor function of MDNN in further sections.

***4.1 MDNN’s nested classes***

The MDNN class comprises of various sub-classes (nested classes) all working and passing information together to compile the MDNN neural network.

There are eight nested classes inside MDNN namely:

1. ActivationsClass
2. LossesClass
3. OptimizersClass
4. AccuracyClass
5. ActivationsFunction
6. LossesFunction
7. OPtimizerFunction
8. Accuracy

Out of which the last four are inherited from above for respectively. The class hierarchy is illustrated in figure 4.1.

MDNN

ActivationsClass

LossesClass

OptimizersClass

AccuracyClass

ActivationsFunction

LossesFunction

OPtimizerFunction

Accuracy

Fig 4.1 Class hierarchy of MDNN

The top four nested class act as type def to define the Enum list of implemented activations, losses, optimizers and accuracy and expose it to user so that it can be used to define what activation of losses user is intending to compile the model. The inherited last four classes that is from 5 to 8 are used to implement the listed functions in Enum of above 1-4 nested class. Since this class contains the implementation hence the classes from 5 to 8 are not exposed and hence it is under private identifier.

The implemented Activations are successively:

1. Relu
2. Sigmoid
3. LeakyReLU
4. Tanh
5. Linear

Whereas implemented losses are:

1. BinaryCrossEntropy
2. CategoricalCrossEntropy

And Optimizers are:

1. SGD
2. Adam
3. RMSProp

***4.2 MDNN’s member functions***

MDNN consists of various crucial and delicate member function all working together to track created model and perform accurate calculation.

The MDNN function are divided in two categories, one which are exposed to user and can be accessed by user, the other which cannot be accessed and modified by user they are delicate and did not mean to be get modified as they undergo matrix calculations.

The public functions, that can be accessed by users are successively:

1. Add
2. Train
3. Summary
4. Input
5. DenseLayer
6. MultidimensionalDenseLayer
7. SaveParameters
8. LoadParameters

And private functions, those which cannot be accessed by users are:

1. add\_to\_record
2. initalize\_input\_neurons
3. initialize\_paramaeters
4. print\_progress\_bar
5. Backpropagation
6. Feedforward
7. Inherited from nested class (5-8) that were been discussed in section 4.1 and implementation of their mathematical logic.

***4.3 MDNN’s member variable***

The MDNN’s member variable act as information exchanger that is, these member variables are accessible throughout the program and are used to carry out the exchange or provide information to the function which is currently it is accessing. There are five member variables, they are:

1. Struct recs
2. Records (Array of structs)
3. RecordIndex
4. parameters (map)
5. InputLayerFlag

***4.4 Documentation***

This document contains the classes defined and the parameters required to access the member function which are accessible to user and it does not contain any logic or algorithm of how the member functions are implemented.

***4.4.1 MDNN Classes***

1. Activations

This class is used to define what activation is needed to be used at any particular level.

Member function and attributes:

|  |  |
| --- | --- |
| Attributes | Definition |
| Relu | Relu activation function |
| Sigmoid | Sigmoid activation function |
| LeakyReLU | LeakyReLU activation function |
| Tanh | Tanh activation function |
| Linear | Linear activation function |

Ex:

//It indicates the MDNN to use sigmoid activation function

MDNN::Activations::Sigmoid

1. **Losses**

This class is used to define what losses will be used to compute the overall loss for training the neural net.

Member function and attributes:

|  |  |
| --- | --- |
| Attributes | Definition |
| BinaryCrossEntropy | Binary cross entropy loss function |
| CategoricalCrossEntropy | Binary cross entropy loss function |

Ex:

//It indicates the MDNN to use BinaryCrossEntropy loss function

MDNN:: Losses:: BinaryCrossEntropy

1. **Optimizers**

This class is used to define which optimizer will be used to optimize the weight gradients.

Member function and attributes:

|  |  |
| --- | --- |
| Attributes | Definition |
| SGD | Stochastic Gradient Descent optimizer |
| Adam | Adaptive moment estimation optimizer |
| RMSProp | Root Mean Squared Propagation optimizer |

Ex:

//It indicates the MDNN to use SGD optimizer

MDNN:: Optimizers:: SGD

1. **Accuracies**

This class tells what accuracy function need to be used to calculate the accuracy of model.

Member function and attributes:

|  |  |
| --- | --- |
| Attributes | Definition |
| Binary\_Accuracy | Binary accuracy |

Ex:

//It indicates the MDNN to use Binary accuracy to calculate accuracies

MDNN::Accuracies:: Binary\_Accuracy

***4.4.2 MDNN member functions***

1. Add

Defined as: void Add(Layer layer)

This attribute function used to add neural layer to model, it takes layer as input.

|  |  |
| --- | --- |
| Parameters | Definition |
| layer | Layer user defined data type, defining the neurons activations and type of neuron |
| Return | Definition |
| None | None |

Ex:

//It indicates the MDNN to add dense layer with 50 neurons to model

Model.Add(Model.DenseLayer(50,MDNN::Activations::Sigmoid));

1. DenseLayer

Defined as:  Layer DenseLayer(int Neurons, ActivationsClass::ActivationFunction ActivationFunction=ActivationsClass::ActivationFunction::Relu)

|  |  |
| --- | --- |
| Parameters | Definition |
| Neurons | Integer number of neurons to add to this layer |
| ActivationFunction | Activation class activation function see section 4.4.1 to add to current layer |
| Return | Definition |
| Layer | It returns the layer type |

Ex:

//It indicates the dense layer with 50 neurons and sigmoid activation function to model

Model.DenseLayer(50,MDNN::Activations::Sigmoid)

1. MultidimensionalDenseLayer

Defined as:  Layer MultidimensionalDenseLayer(int Neurons, int Dimensions, Activations\* ActivationFunctions)

|  |  |
| --- | --- |
| Parameters | Definition |
| Neurons | Integer number of neurons to add to this layer |
| Dimensions | Integer number of dimensions to MDL |
| ActivationFunction | Array of activation class activation function see section 4.4.1 to add to current layer |
| Return | Definition |
| Layer | It returns the layer type |

Ex:

//It defines the array of activation functions to pass as parameters to MDL

MDNN::Activations Activations[2] = {MDNN::Activations::Sigmoid, MDNN::Activations::Tanh};

//It indicates the MDL with 25 neurons and 2 dimension having sigmoid and tanh activation function respectively to model.

        Model.MultidimensionalDenseLayer(25,2,Activations)

1. Input

Defined as:  Layer Input(int Neurons)

|  |  |
| --- | --- |
| Parameters | Definition |
| Neurons | Integer number of neurons to add to this layer |
| Return | Definition |
| Layer | It returns the layer type |

Ex:

//It indicates the input have dimension of Neuron i.e., 2.

    Model.Add(Model.Input(2));

1. Summary

Defined as:  void Summary()

It executes to print the summary of model to console.

|  |  |
| --- | --- |
| Parameters | Definition |
| None | None |
| Return | Definition |
| Null | Null |

Ex:

    Model.Summary();

1. Train

Defined as:   map<string,dmat> Train(arma::field<dmat> input, arma::field<dmat> output, Optimizers optimizer, Losses loss, int epochs, int batch\_size=1)

|  |  |
| --- | --- |
| Parameters | Definition |
| input | dmat field of input dmat matrices |
| output | dmat field of expected outputs or actual outputs of dmat matrices |
| optimizer | Optimizer class optimizer see section 4.4.2 |
| loss | Loss class loss see section 4.4.2 |
| epochs | Integer number of epochs |
| batch\_size | Integer batch size default to 1 |
| Return | Definition |
| map<string,dmat> | It returns the map of dmat matrices of containing   |  |  | | --- | --- | | ‘losses’ | dmat of all losses over epochs | | ‘accuracies’ | dmat of all accuracies over epochs | | ‘ttpe’ | dmat of all time taken per epoch | |

Ex:

//It compiles the model and start training with X\_train as inputs and Y\_train as expected output with SGD optimizer and BinaryCrossEntropy loss for 10000 epochs and the batch of 400 at time. It returns the map of losses, accuracies and ttpe.

map<string,dmat> TRAIN\_OUTPUTS = Model.Train(X\_Train, Y\_Train, MDNN::Optimizers::SGD, MDNN:: Losses:: BinaryCrossEntropy, 10000,400);

1. SaveParameters

Defined as:  void SaveParameters(string filename)

It saves the model weights and biases from parameter map to file

|  |  |
| --- | --- |
| Parameters | Definition |
| filename | Name of file in way you want to save parameter map |
| Return | Definition |
| Null | Null |

Ex:

//It saves the current trained weights to file

    Model.SaveParameters("save.bin");

1. LoadParameters

Defined as:  void LoadParameters(string filename)

It load the model weights and biases from file to parameter map.

|  |  |
| --- | --- |
| Parameters | Definition |
| filename | Filename to load weights and biases from |
| Return | Definition |
| Null | Null |

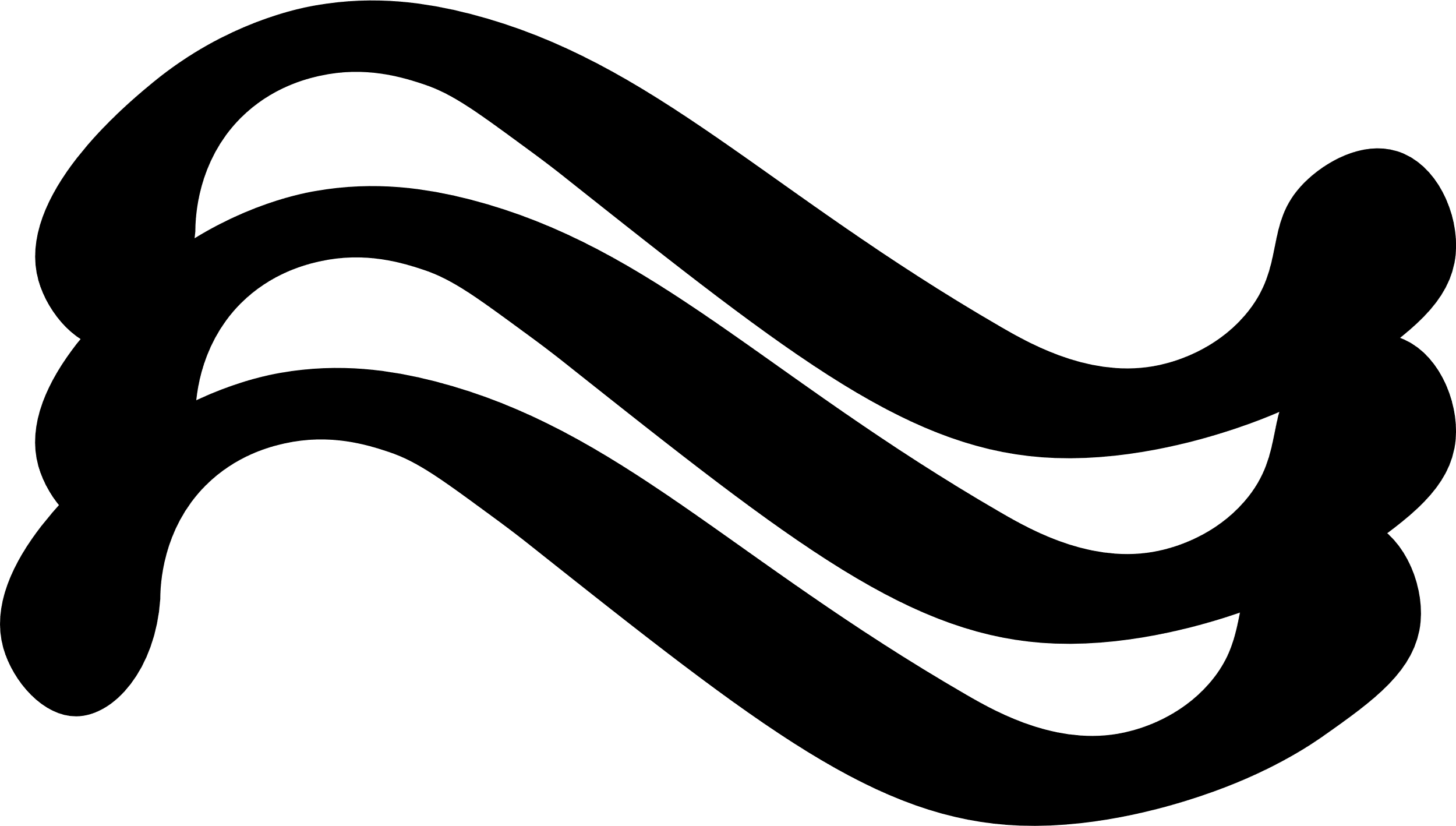
Ex:

//It loads the trained weights from file

    Model.LoadParameters("save.bin");

Chapter 5

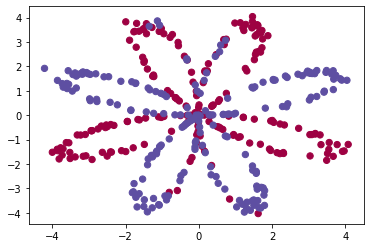
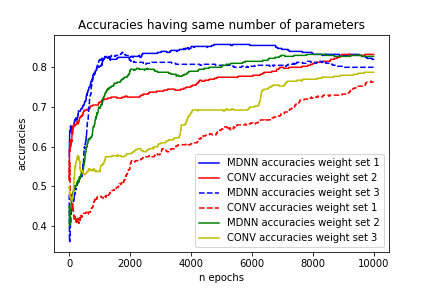
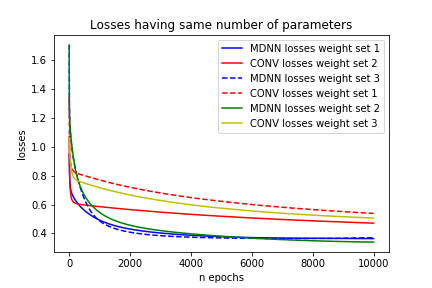
**Experiments and Results**



W

e have experimented MDNN on various datasets e.g., planner dataset classification. The loss convergence was far more in early epoch than that of conventional as well as, MDNN is able to achieve higher accuracies then that of conventional nets. We have experimented planner dataset classification with three different weights set i.e., same weights for both MDNN and conventional networks. Both models have same number of total trainable parameters and exact equal number of neurons at any particular layer. The only difference was of one hidden layer that is MDNN was having multidimensional hidden layer while conventional neural network was having conventional dense layer.

The planner dataset classification experimented with 400 data points, out of which 200 were labelled red and 200 were labelled blue figure 5.1 (A). The goal of model was to classify both categories distinctively. Both models were trained for 10,000 epochs with batch of 400. After training we observe that our proposed architecture MDNN converges losses at much faster than the conventional neural nets, meanwhile also attaining higher accuracy than conventional neural nets and having same average computational time as that of conventional neural nets see figure 5.2.

**B**

**C**

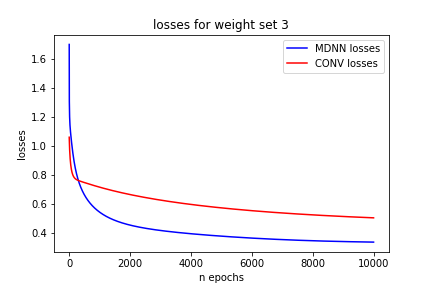
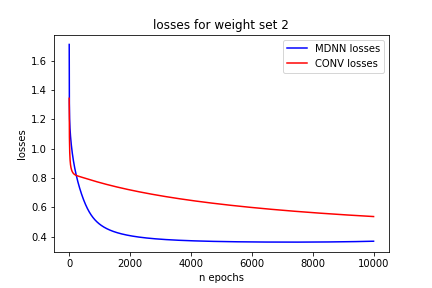
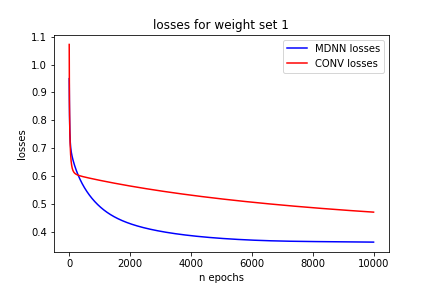
**A**

Fig 5.1. (**A**) Planner dataset consisting 200 red dots and 200 blue dots. (**B**) Losses plot for both mdnn and conv on different weight sets all combines into one. (**C**) Accuracies plot for both mdnn and conv on different weight sets all combines into one.

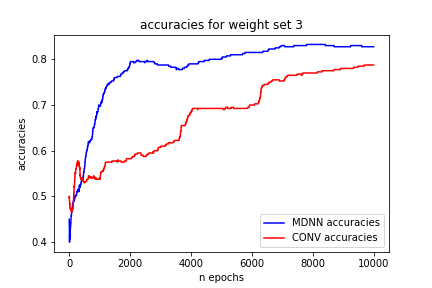
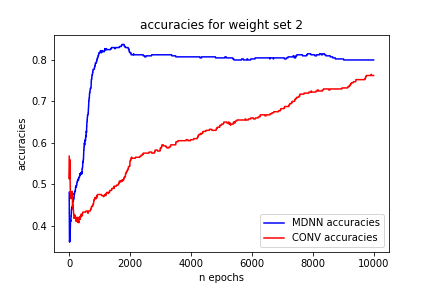
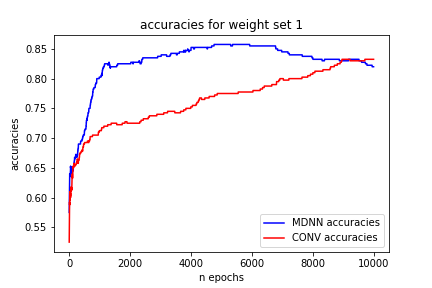
Weight set 3

Weight set 2

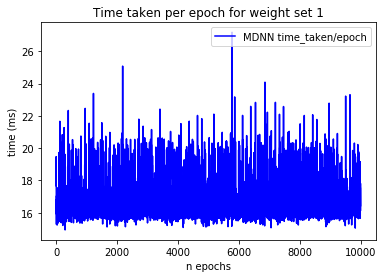
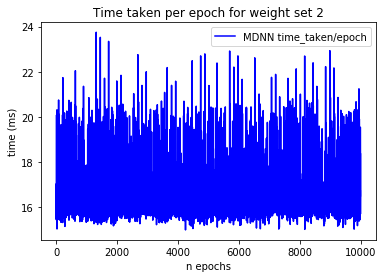
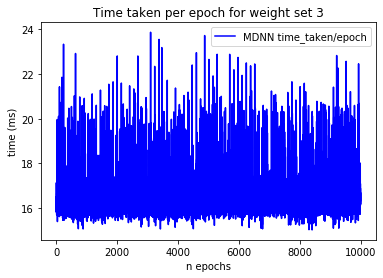
Weight set 1



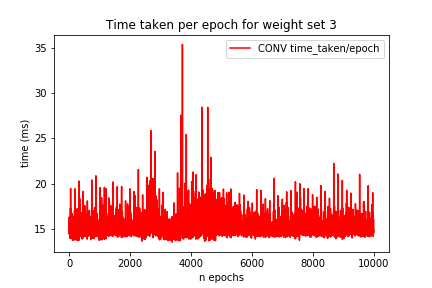
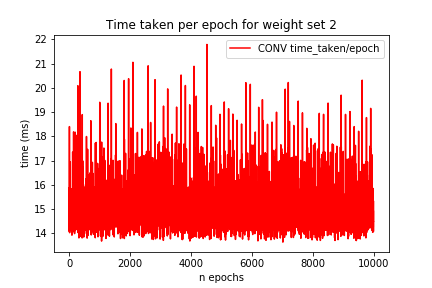
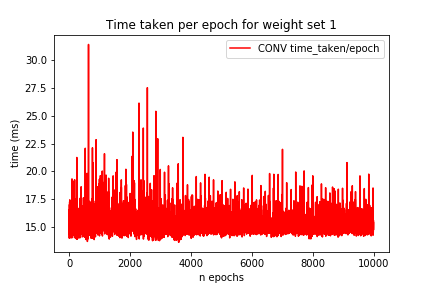
Losses:



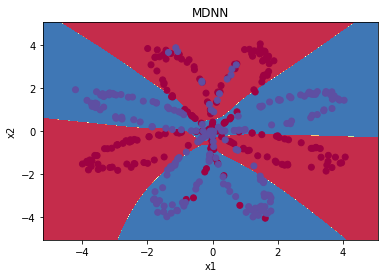
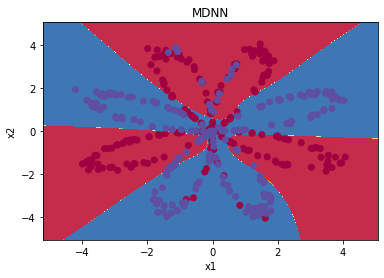
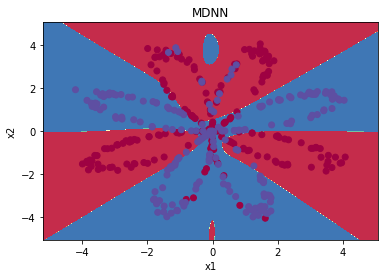
Accuracies:

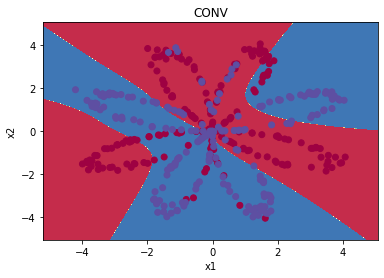
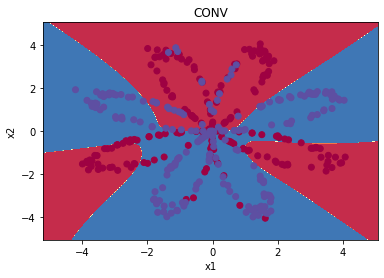
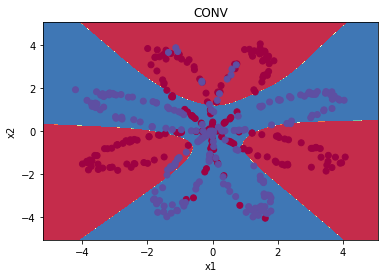
Time taken per epoch (MDNN):



Time taken per epoch (CONV):

Decision boundary (MDNN):

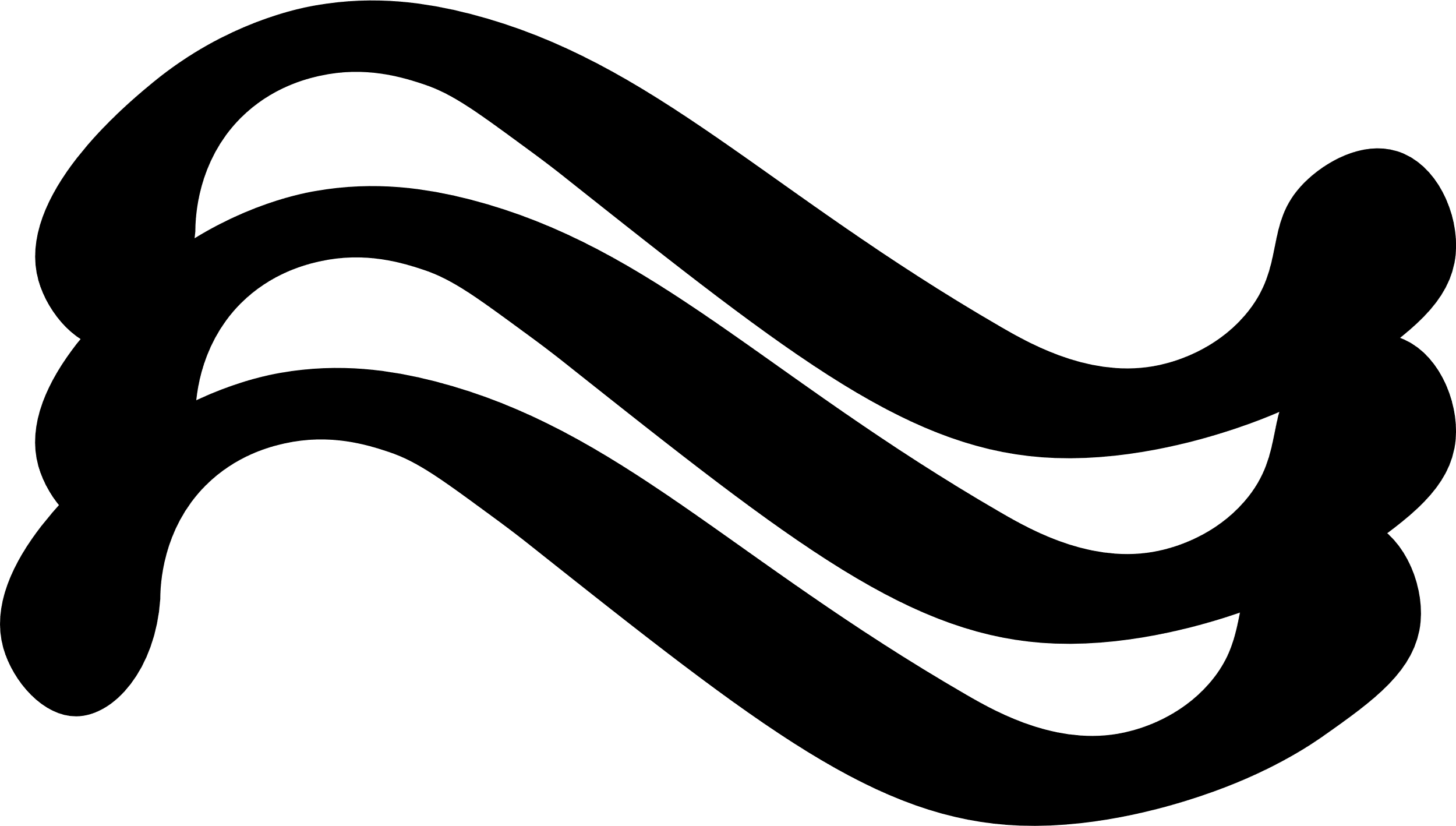
Decision boundary (CONV):

Fig 5.2. Losses, accuracies, Time taken per epoch and Decision boundary plot for MDNN and CONV neural network

From the above figure 5.2 it is clear that MDNN is way better than conventional neural network. As it takes same computational time and attains higher accuracies in little iteration of training. Notice how the losses gets converge more quickly and model attains higher accuracies in less training in MDNN than the conventional neural networks moreover the plot of losses and accuracies for three different weight sets further confirms the consistency of results. Furthermore, the decision boundary plot of MDNN and conventional neural network is convincing that MDNN is a plus to conventional neural networks without compromising with computational time, it is true that it increases little complexity but meanwhile it shows better results.

Chapter 6

**Conclusion And Future Work**



A

s due to the introduction of multi-dimensional layers, allows us in having different activation functions at same level. The use of multiple activation function in MDNN results in better loss convergence and achieving accuracies much faster than conventional neural network. By using the proper combination of activation function to use in pairs it would further boost the performance of MDNN. Although, it was thought that the introduction of new dimensionality feature would cost the computational time but the results were otherwise, the computation time was exactly similar to its counter conventional neural network provided both have same number of neurons at any particular layer. Which further shows that MDNN are superior than conventional neural networks.

We have shown that by the use of MDNN, any neural network would boost its performance. As MDNN allows the creator a flexibility of the dimensionality parameter as we talked earlier. This could be used to introduce multi-activated dense layer in conventional neural networks meantime without compromising the cost of computational timings. Our future work includes of introducing the interconnectivity between the dimensions of multi-dimensional layer i.e., a dense mapping between the dimensions of having different activation functions and calculating its gradients and optimizing it appropriately.

Appendix



***A.1 Example program for Planner data classification***

int main(int argc, char\*\* argv){

    MDNN Model;

    arma::field<dmat> X\_Train;

    arma::field<dmat> Y\_Train;

    X\_Train.load("X\_train\_field.bin");

    Y\_Train.load("Y\_train\_field.bin");

    int MODE=0;

    cout<<"0. MDNN\n1. CONV. \n";

    cin>>MODE;

    Model.Add(Model.Input(2));

    MDNN::Activations Activations[2]={MDNN::Activations::Sigmoid,MDNN::Activations::Tanh};

    if(MODE==0){

        Model.Add(Model.MultidimensionalDenseLayer(25,2,Activations));

    }

    else{

        Model.Add(Model.DenseLayer(50,MDNN::Activations::Sigmoid));

    }

    Model.Add(Model.DenseLayer(1,MDNN::Activations::Sigmoid));

    Model.Summary();

    string weight\_set = "Weights\_set\_3/";

    string whose = "\_planner\_dataset\_E3\_";

    if(MODE==0){

        whose = whose+"MDNN";

        Model.LoadParameters(weight\_set+"MDNN/"+"save\_MDNN\_3.bin");

    }

    else{

        whose = whose+"CONV";

        Model.LoadParameters(weight\_set+"CONV/"+"save\_CONV\_3.bin");

    }

    map<string,dmat> TRAIN\_OUTPUTS = Model.Train(X\_Train,Y\_Train,MDNN::Optimizers::SGD,10000,400);

    bool lossSave = TRAIN\_OUTPUTS["losses"].save("losses"+whose+".csv",arma::csv\_ascii);

    if(lossSave)cout<<"Lossess saved sucessfull."<<endl;

    bool accuracySave = TRAIN\_OUTPUTS["accuracies"].save("accuracies"+whose+".csv",arma::csv\_ascii);

    if(accuracySave)cout<<"Accuracies saved sucessfull."<<endl;

    bool timeSave = TRAIN\_OUTPUTS["ttpe"].save("ttpe"+whose+".csv",arma::csv\_ascii);

    if(timeSave)cout<<"Time\_taken\_per\_epoch saved sucessfull."<<endl;

    return 0;

}

***A.2*** Statements and Declarations

***A.2.1 Funding***

We wish to confirm that there has been no significant financial support for this work that could have influenced its outcome.

***A.2.2 Conflict of interest***

We wish to confirm that there are no known conflicts of interest associated with this publication.

***A.2.3 Ethics approval***

We further confirm that the work covered in this manuscript has not involved either experimental animals or human patients. Ethics approval not applicable to this research.

***A.2.4 Consent to participate***

We confirm that the manuscript has been read and approved by all named authors and that there are no other persons who satisfied the criteria for authorship but are not listed.

A.2.5 Consent for publication

We confirm that there’s no impediments for publishing the submitted manuscript.

A.2.6 Code availability

We confirm that corresponding codes and the dataset generated for the work done in this manuscript are available from the corresponding author on reasonable request.

A.2.7 Authors’ contributions

We confirm that contribution to the paper as follows:

**Idea, study conception and model design**: Venish Patidar,

**Math calculations and code implementation**: Venish Patidar,

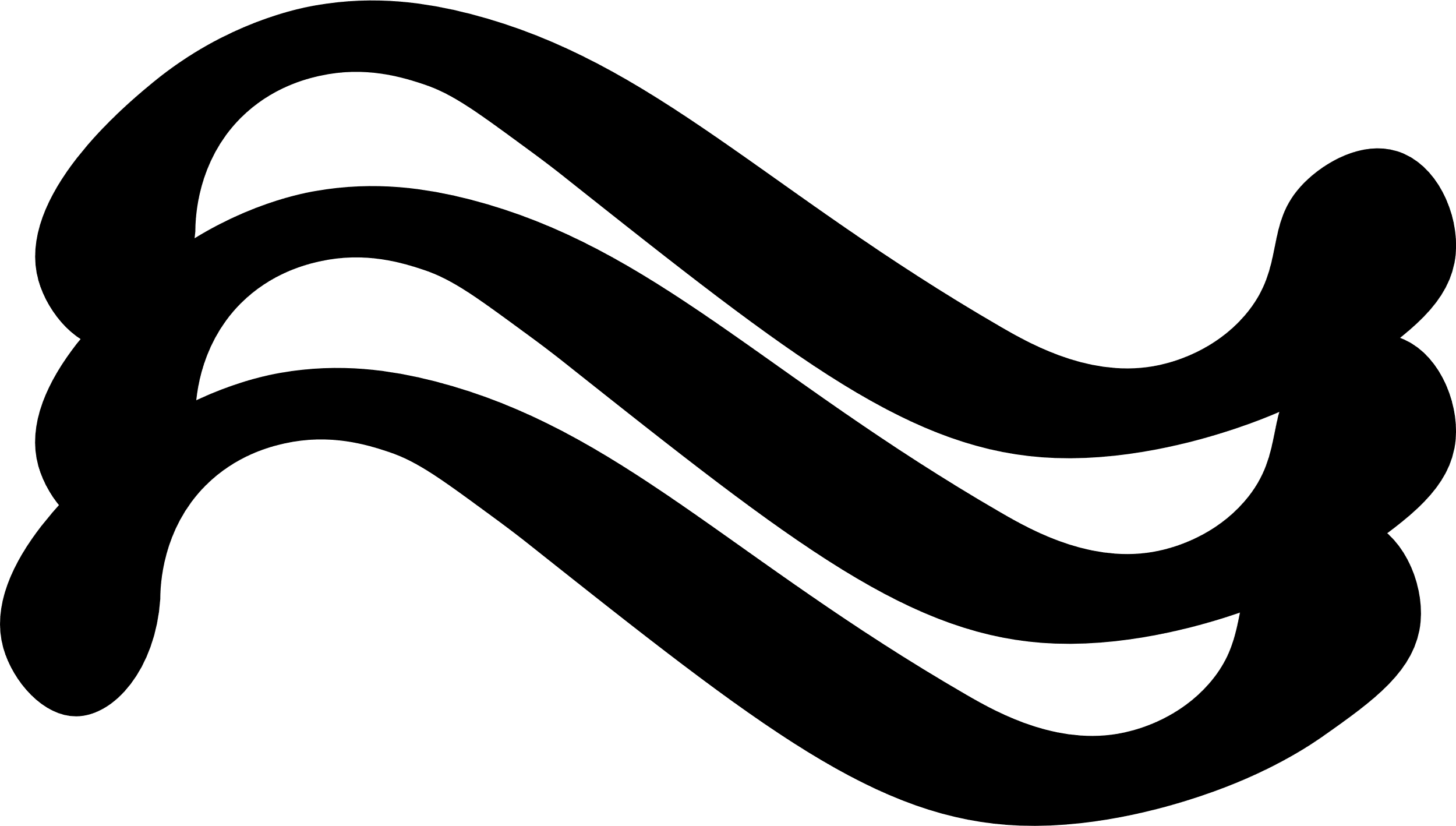
**Debugging the program**: Venish Patidar, Raj Gupta, Yash Biyani,

**Finished work revision**: Venish Patidar, Raj Gupta, Yash Biyani, Prof. Preetesh Purohit,

**Manuscript preparation and revision**: Venish Patidar, Raj Gupta, Yash Biyani, Prof. Preetesh Purohit.

We understand that the Corresponding Author is the sole contact for the Editorial process. He is responsible for communicating with the other authors about progress, submissions of revisions and final approval of proofs. We confirm that we have provided a current, correct email address which is accessible by the Corresponding Author.

**Bibliography**



[1] Charu C. Aggarwal, “Neural Networks and Deep Learning: A Textbook”, September 2018, ch.1, sec.1.2, page 4-17

[2] Charu C. Aggarwal, “Neural Networks and Deep Learning: A Textbook”, September 2018, ch.1, sec.1.2.2, page 17-20

[3] Weibo Liu, et al, “A survey of deep neural network architectures and their applications”, vol. 234, 2017, page 11-26, ISSN 0925-2312, 10.1016/j.neucom.2016.12.038.

[4] Z. -Q. Zhao, P. Zheng, S. -T. Xu and X. Wu, “Object Detection with Deep Learning: A Review”, in IEEE Transactions on Neural Networks and Learning Systems, vol. 30, no. 11, pp. 3212-3232, Nov. 2019, Doi: 10.1109/TNNLS.2018.2876865.

[5] Inaguma, Hirofumi; Duh, Kevin; Kawahara, Tatsuya; Watanabe, Shinji (2019), IEEE Automatic Speech Recognition and Understanding Workshop (ASRU) - Multilingual End-to-End Speech Translation, 570–577, doi:10.1109/ASRU46091.2019.9003832

[6] Bharath Ramsundar, Reza Bosagh Zadeh, TensorFlow for Deep Learning, O'Reilly, March 2018 fully connected deep network, <https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html>

[7] Wikipedia contributors. (2021, May 5). Layer (deep learning). In Wikipedia, <https://en.wikipedia.org/w/index.php?title=Layer_(deep_learning)&oldid=1021579399>

[8] Goodfellow et al, “Deep Learning”, MIT Press, 2016 <http://www.deeplearningbook.org>, part 2, Deep Feedforward Networks

[9] Charu C. Aggarwal, “Neural Networks and Deep Learning: A Textbook”, September 2018, ch.1, sec.1.3, page 21

[10] Jason Brownlee, “Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions”, December 2018, ch.4, sec.4.1, “Loss Functions”, page 49

[11] Jeremy Jordan, “Neural networks: training with backpropagation” data science,18 July,2017<https://www.jeremyjordan.me/neural-networks-training>

[12] Dami Choi et al, “On Empirical Comparisons of Optimizers for Deep Learning”, arXiv:1910.05446