MET Bhujbal Knowledge City
Institute of Engineering,
Nashik.

Department of AI & DS Engineering
2023-24

Lab Manual
Software Laboratory II
(317533)
(Artificial Neural Network)
TE-SEM-II

Prepared By:
Prof. Rahul B. Mandlik

HOD
Dr. P.M. Yawalkar

Course Objectives:

- To understand basic techniques and strategies of learning algorithms
- To understand various artificial neural network models
- To make use of tools to solve the practical problems in real field using Pattern Recognition, Classification and Optimization.

Course Outcomes:

On completion of the course, learner will be able to–

CO1: Model artificial Neural Network, and to analyze ANN learning, and its applications
CO2: Perform Pattern Recognition, Linear classification.
CO3: Develop different single layer/multiple layer Perception learning algorithms
CO4: Design and develop applications using neural networks.

Guidelines for Instructor's Manual

The instructor‘s manual is to be developed as a hands-on resource and reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of course, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm,flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable),conclusion/analysis. Program codes with sample output of all performed assignments are to be submitted as softcopy.
As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student’s for advanced learning, understanding of the fundamentals, effective and efficient implementation. So encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising start of the student's academics.

INDEX

Course Structure

Savitribai Phule Pune University

Third Year of Artificial Intelligence and Data Science (2019 Course)

(With effect from Academic Year 2022-23)

Semester-VI

Course Code	Course Name	Teaching Scheme ##(Hours/Week)			Examination Scheme and Marks						Credit Scheme			
		#Lecture	Practical	Tutorial	Mid-Sem	End-Sem	Term work	Practical	Oral	Total	Lecture	Practical	Tutorial	Total
317529	Data Science	04	-	-	30	70	-	-	-	100	03	--	-	03
317530	Cyber security	04	-	-	30	70	-	-	-	100	03	-	-	03
317531	Artificial Neural Network	04	-	-	30	70	-	-	-	100	03	-	-	03
**	Elective II	04	-	-	30	70	-	-	-	100	03	-	-	03
317533	Software Laboratory II	-	04	-	-	-	25	25	-	50	-	02	-	02
317534	Software Laboratory III	-	04	-	-	-	50	25	-	75	-	02	-	02
317535	Internship**	-	--	-	-	-	50	-	50	100	-	04	-	04
317536	**Mini Project (CS and Elective-II)**	-	02	-	-	-	50	-	25	75	-	01	-	01
	Total	**16**	**10**	**-**	**120**	**280**	**175**	**50**	**75**	**700**	**12**	**09**	**-**	**21**
317537	Audit Course 6										**Grade**			
									Total		**12**	**09**	**-**	**21**

Savitribai Phule Pune University
Third Year of Artificial Intelligence and Data Science (2019 Course)
317533: Software Laboratory II

Teaching Scheme:	Credit	Examination Scheme:
PR: 04 Hours/Week	02	Term Work (TW): 25 Marks Practical(PR): 25 Marks

Prerequisite Courses, if any: **Software Laboratory I (317526),Elective I Laboratory (317525)**

Companion Course, if any: **Artificial Neural Network (317534)**

Course Objectives:

- To understand basic techniques and strategies of learning algorithms
- To understand various artificial neural network models
- To make use of tools to solve the practical problems in real field using Pattern Recognition, Classification and Optimization

Course Outcomes:

On completion of the course, learner will be able to–

CO1: Model artificial Neural Network, and to analyze ANN learning, and its applications
CO2: Perform Pattern Recognition, Linear classification.
CO3: Develop different single layer/multiple layer Perception learning algorithms
CO4: Design and develop applications using neural networks.

Guidelines for Instructor's Manual

The instructor‘s manual is to be developed as a hands-on resource and reference. The instructor's manual need to include prologue (about University/program/ institute/ department/foreword/ preface), curriculum of course, conduction and Assessment guidelines, topics under consideration-concept, objectives, outcomes, set of typical applications/assignments/ guidelines, and references

Guidelines for Student's Laboratory Journal

The laboratory assignments are to be submitted by student in the form of journal. Journal consists of prologue, Certificate, table of contents, and **handwritten write-up**of each assignment (Title, Objectives, Problem Statement, Outcomes, software and Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory- Concept in brief, algorithm,flowchart, test cases, Test Data Set(if applicable), mathematical model (if applicable),conclusion/analysis. **Program codes with sample output of all performed assignments are to besubmitted as softcopy.**

As a conscious effort and little contribution towards Green IT and environment awareness, attaching printed papers as part of write-ups and program listing to journal may be avoided. Use of DVD containing students programs maintained by Laboratory In-charge is highly encouraged. For reference one or two journals may be maintained with program prints at Laboratory.

Guidelines for Laboratory / Term Work Assessment

Continuous assessment of laboratory work should be done based on overall performance and Laboratory assignments performance of student. Each Laboratory assignment assessment should be assigned grade/marks based on parameters with appropriate weightage. Suggested parameters for overall assessment as well as each Laboratory assignment assessment include- timely completion, performance, innovation, efficient codes, punctuality and neatness.

Guidelines for Laboratory Conduction

The instructor is expected to frame the assignments by understanding the prerequisites, technological aspects, utility and recent trends related to the topic. The assignment framing policy need to address the average students and inclusive of an element to attract and promote the intelligent students. The instructor may set multiple sets of assignments and distribute among batches of students. It is appreciated if the assignments are based on real world problems/applications. Encourage students for appropriate use of Hungarian notation, proper indentation and comments. Use of open source software is to be encouraged. In addition to these, instructor may assign one real life application in the form of a mini-project based on the concepts learned. Instructor may also set one assignment or mini-project that is suitable to respective branch beyond the scope of syllabus.

Set of suggested assignment list is provided in groups- A, B, and C. Each student must perform at least

Curriculum for Third Year of Artificial Intelligence and Data Science (2019 Course), Savitribai Phule Pune University

10 assignments and one mini project (at least 6 from group A, 2 from group B and 2 from group C)

Group A and B assignments should be implemented in Python without using built-in methods for major functionality of assignment. Operating System recommended:- 64-bit Open source Linux or its derivative Programming tools recommended: - Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder, Tensorflow.

Guidelines for Practical Examination

Both internal and external examiners should jointly set problem statements. During practical assessment, the expert evaluator should give the maximum weightage to the satisfactory implementation of the problem statement. The supplementary and relevant questions may be asked at the time of evaluation to test the student's for advanced learning, understanding of the fundamentals, effective and efficient implementation. So encouraging efforts, transparent evaluation and fair approach of the evaluator will not create any uncertainty or doubt in the minds of the students. So adhering to these principles will consummate our team efforts to the promising start of the student's academics.

Virtual Laboratory:

https://cse22-iiith.vlabs.ac.in/

http://vlabs.iitb.ac.in/vlabs-dev/labs/machine_learning/labs/index.php

Suggested List of Laboratory Experiments/Assignments

Group A (Any 6)

1. Write a Python program to plot a few activation functions that are being used in neural networks.
2. Generate ANDNOT function using McCulloch-Pitts neural net by a python program.
3. Write a Python Program using Perceptron Neural Network to recognise even and odd numbers. Given numbers are in ASCII form 0 to 9
4. With a suitable example demonstrate the perceptron learning law with its decision regions using python. Give the output in graphical form.
5. Write a python Program for Bidirectional Associative Memory with two pairs of vectors.
6. Write a python program to recognize the number 0, 1, 2, 39. A 5 * 3 matrix forms the numbers. For any valid point it is taken as 1 and invalid point it is taken as 0. The net has to be trained to recognize all the numbers and when the test data is given, the network has to recognize the particular numbers
7. Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.
8. Create a Neural network architecture from scratch in Python and use it to do multi-class classification on any data.

   Parameters to be considered while creating the neural network from scratch are specified as:

   (1) No of hidden layers : 1 or more

   (2) No. of neurons in hidden layer: 100

   (3) Non-linearity in the layer : Relu

   (4) Use more than 1 neuron in the output layer. Use a suitable threshold value

   Use appropriate Optimisation algorithm

Group B (Any 4)

1. Write a python program to show Back Propagation Network for XOR function with Binary Input and Output
2. Write a python program to illustrate ART neural network.
3. Write a python program in python program for creating a Back Propagation Feed-forward neural network
4. Write a python program to design a Hopfield Network which stores 4 vectors
5. Write Python program to implement CNN object detection. Discuss numerous performance evaluation metrics for evaluating the object detecting algorithms' performance.

Group C (Any 3)

1. How to Train a Neural Network with TensorFlow/Pytorch and evaluation of logistic regression using tensorflow

2. TensorFlow/Pytorch implementation of CNN

3. For an image classification challenge, create and train a ConvNet in Python using TensorFlow. Also try to improve the performance of the model by applying various hyper parameter tuning to reduce the overfitting or under fitting problem that might occur. Maintain graphs of comparisons.

4. MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Mini Project

Car Object Detection using (ConvNet/CNN) Neural Network
Car Object Data: Data Source – https://www.kaggle.com/datasets/sshikamaru/car-object-detection
The dataset contains images of cars in all views.
Training Images – Set of 1000 files
Use Tensorflow, Keras & Residual Network resNet50
Constructs comparative outputs for various Optimisation algorithms and finds out good accuracy.

OR

Mini Project to implement CNN object detection on any data. Discuss numerous performance evaluation metrics for evaluating the object detecting algorithms' performance, Take outputs as a comparative results of algorithms.

Learning Resources

Text Books:

1. Neural Networks a Comprehensive Foundations, Simon Haykin, PHI edition.
2. Laurene Fausett:Fundamentals of Neural Networks: Architectures, Algorithms & Apps, Pearson, 2004.
3. Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python 1st ed. Edition, Apress publication

Reference Books:

1. Getting Started with TensorFlow, by Giancarlo Zaccone
2. AI and Machine learning for coders by Laurence Moroney, O'Reilly Media, Inc.

e-Books:

1. https://www.inf.ed.ac.uk/teaching/courses/nlu/assets/reading/Gurney_et_al.pdf
2. http://neuralnetworksanddeeplearning.com/

MOOC Courses:

1. **http://neuralnetworksanddeeplearning.com/**
2. **https://www.coursera.org/learn/convolutional-neural-networks-tensorflow**
3. **https://nptel.ac.in/courses/106106213**

@The CO-PO mapping table

PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2		2								2
CO2	1	2		2								2
CO3	2	2	2									2
CO4	2	2	2	2								2

MET Bhujbal Knowledge City, Institute of Engineering, Nashik.
DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

CO-PO MAPPING

@The CO-PO mapping table												
PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	2		2								2
CO2	1	2		2								2
CO3	2	2	2									2
CO4	2	2	2	2								2

EXPERIMENT NO. 1 (Group A)

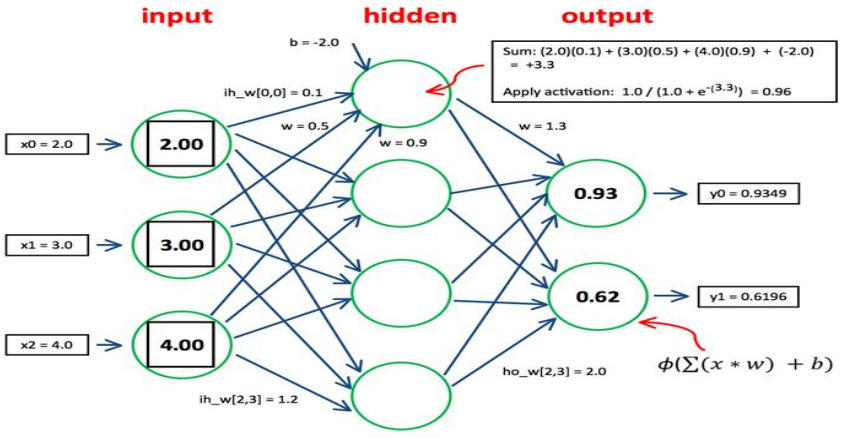
Aim: Write a Python program to plot a few activation functions that are being usedin neural networks.

Outcome: At end of this experiment, student will be able to write a python programto plot a few activation functions that are being used in neural networks.
Hardware Requirement:

Software Requirement: Ubuntu OS,Python Editor(Python Interpreter)

Theory:
In the process of building a neural network, one of the choices you get to make is what Activation Function to use in the hidden layer as well as at the output layer of the network.
Elements of a Neural Network
Input Layer: This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.
Hidden Layer: Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.
Output Layer: This layer bring up the information learned by the network to the outer world.

What is an activation function and why use them?
The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activationfunction is to introduce non-linearity into the output of a neuron.
Activation functions make the back- propagation possible since the gradients are supplied along with the error to update the weights and biases.
Why do we need Non-linear activation function?
A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.
Mathematical proof
Suppose we have a Neural net like this :-



Elements of the diagram are as follows:
Hidden layer i.e. layer 1:
z(1) = W(1)X + b(1) a(1)
Here,

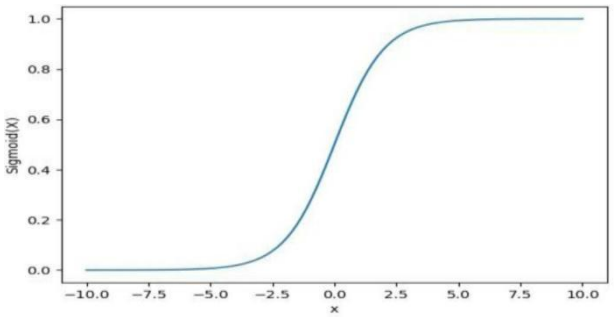
- z(1) is the vectorized output of layer 1
- W(1) be the vectorized weights assigned to neurons of hidden layer i.e. *w1, w2, w3 and w4*
- X be the vectorized input features i.e. *i1 and i2*
- b is the vectorized bias assigned to neurons in hidden layer i.e. *b1 and b2*
- a(1) is the vectorized form of any linear function.

(Note: We are not considering activation function here)
Layer 2 i.e. output layer :-
Note : Input for layer 2 is output from layer 1
z(2) = W(2)a(1) + b(2)
a(2) = z(2)
Calculation at Output layer
z(2) = (W(2) * [W(1)X + b(1)]) + b(2)
z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]
Let,
[W(2) * W(1)] = W
[W(2)*b(1) + b(2)] = b
Final output : z(2) = W*X + b which is again a linear function
This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because *the composition of two linear function is a linear function itself*. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. Hence we need an activation function.
Variants of Activation Function
Linear Function

- Equation : Linear function has the equation similar to as of a straight line i.e. y = x
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- Range : -inf to +inf
- Uses : Linear activation function is used at just one place i.e. output layer.

For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer.

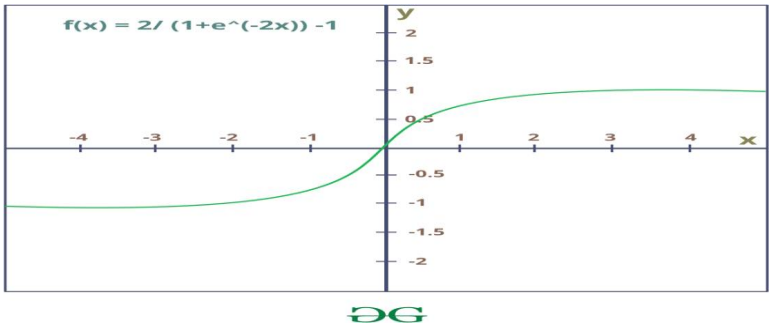
Sigmoid Function:



- It is a function which is plotted as 'S' shaped graph.
- Equation : **$A = 1/(1 + e^{-x})$**
- Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.

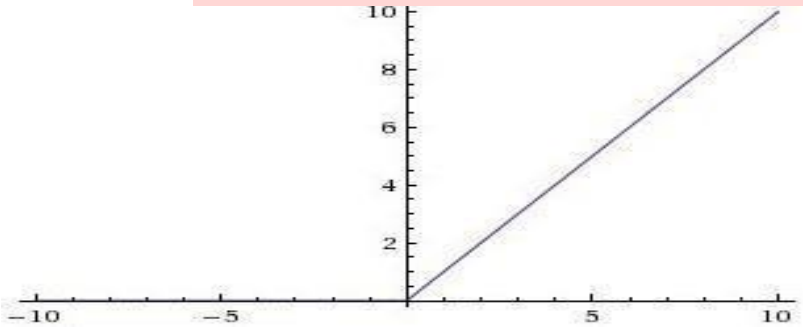
- Value Range : 0 to 1
- Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be *1* if value is greater than 0.5 and *0* otherwise.

Tanh Function



- The activation that works almost always better than sigmoid function is Tanh function also known as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- Equation :- $f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$
- Value Range :- -1 to +1
- Nature :- non-linear
- Uses: - Usually used in hidden layers of a neural network as its values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Function:

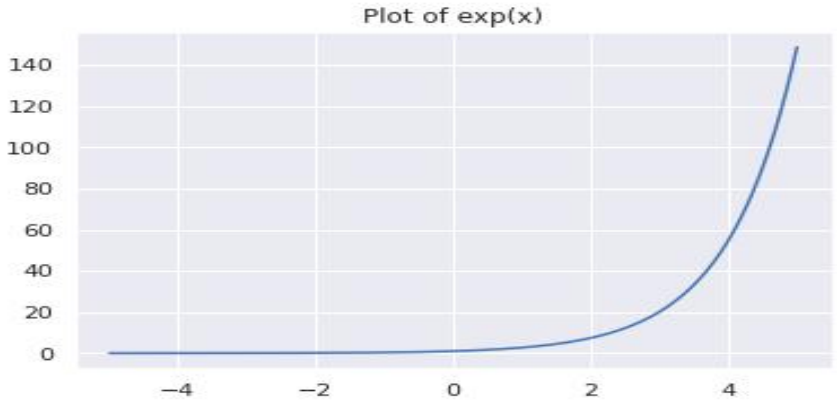


- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- Equation :- *A(x) = max(0,x)*. It gives an output x if x is positive and 0 otherwise.
- Value Range :- [0, inf)
- Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

Softmax Function:

`



Plot of exp(x)

The softmax function is also a type of sigmoid function but is handy when we are tryingto handle multi- class classification problems.

- Nature :- non-linear
- Uses :- Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- Output:- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Conclusion: -

_____

_____

_____

Questions:

Q1. What is the role of the Activation functions in Neural Networks?
Q2. List down the names of some popular Activation Functions used in Neural Networks?
Q3. How to initialize Weights and Biases in Neural Networks?
Q4. How are *Neural Networks* modelled?
Q5. What is an *Activation Function*?

EXPERIMENT NO. 2 (Group A)

Aim: Generate ANDNOT function using McCulloch-Pitts neural net by a python program.

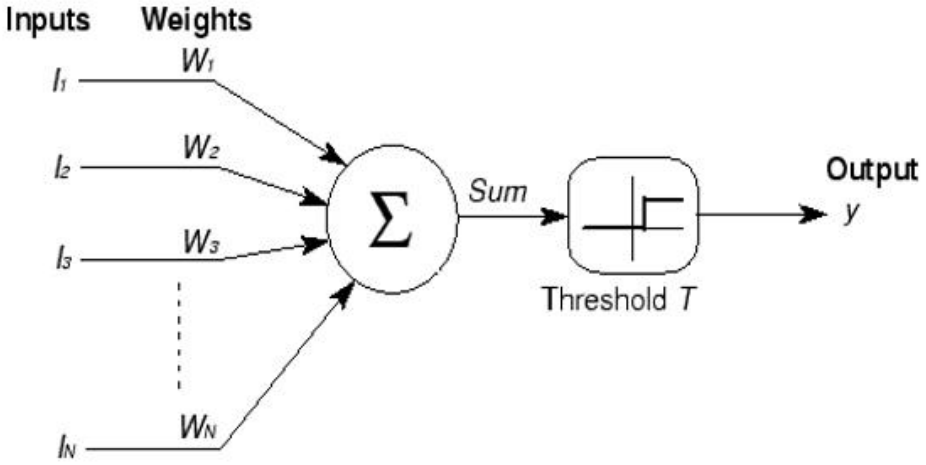
Outcome: At end of this experiment, student will be able to Generate ANDNOTfunction using McCulloch-Pitts neural net by a python program.

Hardware Requirement:

Software Requirement: Ubuntu OS,Python Editor(Pyhton Interpreter)

Theory:

The early model of an artificial neuron is introduced by Warren McCulloch and Walter Pitts in 1943. The McCulloch-Pitts neural model is also known as linear threshold gate. It is a neuron of a set of inputs and one output . The linear threshold gate simply classifies the set of inputs into two different classes. Thus the output is binary. Such afunction can be described mathematically using these equations: W1,w2,w3 are weight values normalized in the range of either or and associated with each input line, is the weighted sum, and is a threshold constant. The function is a linear step function at threshold as shown in figure below. The symbolic representation of the linear threshold gate is shown in figure.



The McCulloch-Pitts model of a neuron is simple yet has substantial computing potential. It also has a precise mathematical definition. However, this model is so simplistic that it only generates a binary output and also the weight and threshold values are fixed. The neural computing algorithm has diverse features for various applications. Thus, we need to obtain the neural model with more flexible computational features.

MLP Classifier trains iteratively since at each time step the partial derivative so the loss functions with respect to the model parameters are computed to update the parameters. It canalso have a regularization term added to the loss function that shrinks model parameters to prevent over fitting.

This implementation works with data represented as dense numpy arrays or sparse scipy arrays of floating point values. Remove error values in the data set and train the modelaccordingly via various operations. To perform classification on data set and predict the outcomes, alsoto plot the outcomes of the experiment to get better understanding of the outcomes.

Truth Table of AND NOT Function

x1	x2	Y
1	1	0
1	0	1
0	1	0
0	0	0

After that, we have to assume two weights w1 = w2 = 1 and w1 = 1, w2 = -1 for the inputs.
A NN with two input neurons and a single output neuron can operate as an ANDNOT logic function if we choose weights
W1 = 1, W2 = -1 and threshold Θ = 1.
Yin is a activation value
X1=1, X2=1,
Yin= W1*X1+W2*X2=1*1+(-1)*1=0, Yin< Θ , so Y=0
X1=1, X2=0
Yin=1*1+0*(-1)=1, Yin= Θ , so Y=1
X1=0, X=1
Yin=0-1=-1, Yin< Θ , so Y=0
X1=0, X2=0
Yin=0, Yin< Θ , so Y=0
So , Y=[0 1 0 0]

Conclusion: -

Questions:
Q1. Explain MCP Model?
Q2. How to generate AND NOT function using MCP Model ?
Q3. Discuss briefly McCulloch Pitt's artificial neuron model.
Q4. Give McCulloch Pitt's artificial neuron model limitations?

EXPERIMENT NO. 3 (Group A)

Aim: Write a Python Program using Perceptron Neural Network to recognize evenand odd numbers. Given numbers are in ASCII from 0 to 9

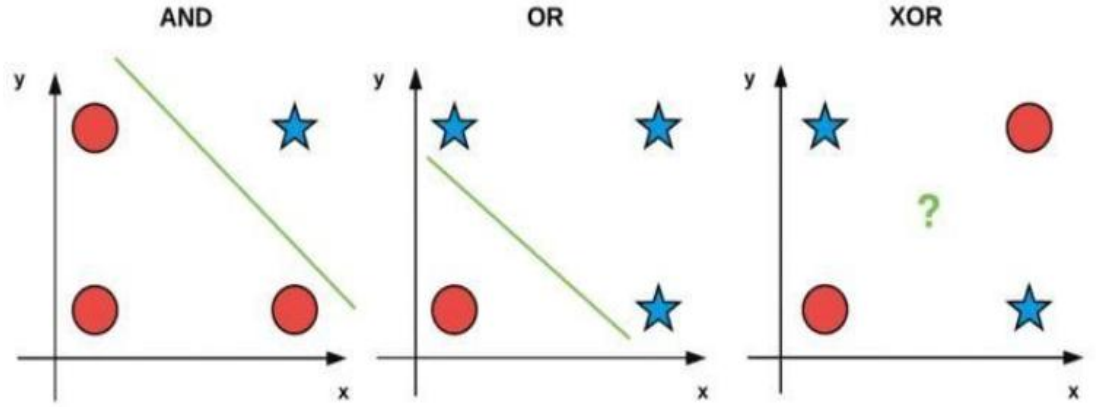
Outcome: At end of this experiment, student will be able to Write a Python Program using Perceptron Neural Network to recognize even and odd numbers.

Hardware Requirement:

Software Requirement: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder, Tensorflow.

Theory:

First introduced by Rosenblatt in 1958, The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain is arguably the oldest and most simple of the ANN algorithms. Following this publication, Perceptron-based techniques were all the rage in the neural network community. This paper alone is hugely responsible for the popularity and utility of neural networks today.



But then, in 1969, an "AI Winter" descended on the machine learning community that almost froze out neural networks for good. Minsky and Papert published Perceptrons: an introduction to computational geometry, a book that effectively stagnated research in neural networks for almost a decade — there is much controversy regarding the book (Olazaran, 1996), but the authors did successfully demonstrate that a single layer Perceptron is unable to separate nonlinear data points.

Given that most real-world datasets are naturally nonlinearly separable, this it seemed that the Perceptron, along with the rest of neural network research, might reach an untimely end.
Between the Minsky and Papert publication and the broken promises of neural networks revolutionizing industry, the interest in neural networks dwindled substantially. It wasn't until we started exploring deeper networks (sometimes called multi-layer perceptrons) along with the backpropagation algorithm (Werbos and Rumelhart et al.) that the "AI Winter" in the 1970s ended and neural network research started to heat up again.

Perceptron Architecture
Rosenblatt (1958) defined a Perceptron as a system that learns using labeled examples (i.e., supervised learning) of feature vectors (or raw pixel intensities), mapping these inputs to their corresponding output class labels.
In its simplest form, a Perceptron contains N input nodes, one for each entry in the input row of the design matrix, followed by only one layer in the network with just a single node in that layer (Figure 2).

There exist connections and their corresponding weights w1, w2, …, wi from the input xi's to the single output node in the network. This node takes the weighted sum of inputs and applies a step function to determine the output class label. The Perceptron outputs either a 0 or a 1 — 0 for class #1 and 1 for class #2; thus, in its original form, the Perceptron is simply a binary, two-class classifier.

1. Initialize our weight vector $\boldsymbol{w}$ with small random values
2. Until Perceptron converges:
(a) Loop over each feature vector x_j and true class label d_i in our training set D
(b) Take $\boldsymbol{x}$ and pass it through the network, calculating the output value: $y_j = f(\boldsymbol{w}(t) \cdot \boldsymbol{x_j})$
(c) Update the weights $\boldsymbol{w}$: $w_i(t+1) = w_i(t) + \alpha(d_j - y_j)x_{j,i}$ for all features $0 <= i <= n$

Figure 3: The Perceptron algorithm training procedure.

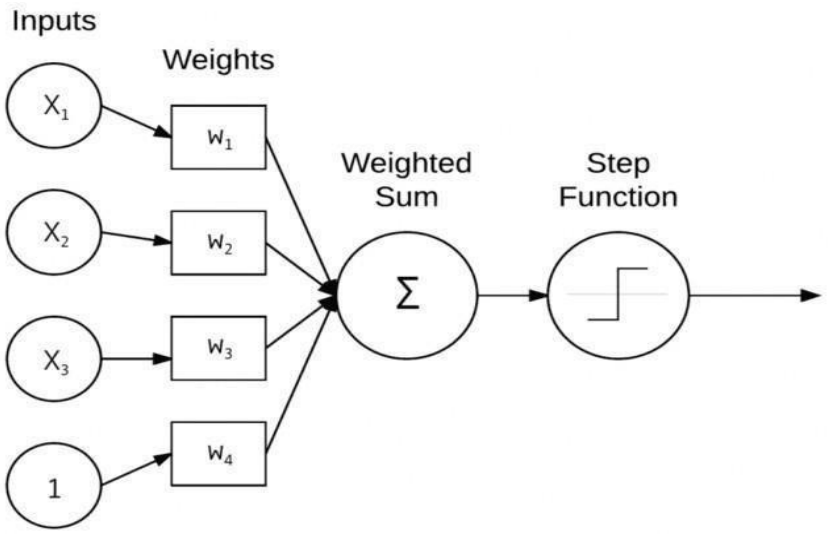


Figure 2: Architecture of the Perceptron network.

NN with two input neurons (one being the variable Number, the other being a bias neuron), nine neurons in the hidden layer and one output neuron using a very simple genetic algorithm: at each epoch, two sets of weights "fight" against each other; the one with the highest error loses and it's replaced by a modified version of the winner.
The script easily solve simple problems like the AND, the OR and the XOR operators but get stuck while trying to categorize odd and even numbers. Right now the best it managed to do is toidentify 53 numbers out of 100 and it took several hours.

Conclusion: -

_____

_____

Questions:

Q1. Is it possible to train a NN to distinguish between odd and even numbers only using as input the numbers themselves?
Q2. Can Perceptron Generalize Non-linear Problems?
Q3. How to create a Multilayer Perceptron NN?
Q4. Is the multilayer perceptron (MLP) a deep learning method? Explain it?

EXPERIMENT NO. 4 (Group A)

Aim: With a suitable example demonstrate the perceptron learning law with itsdecision regions using python. Give the output in graphical form.

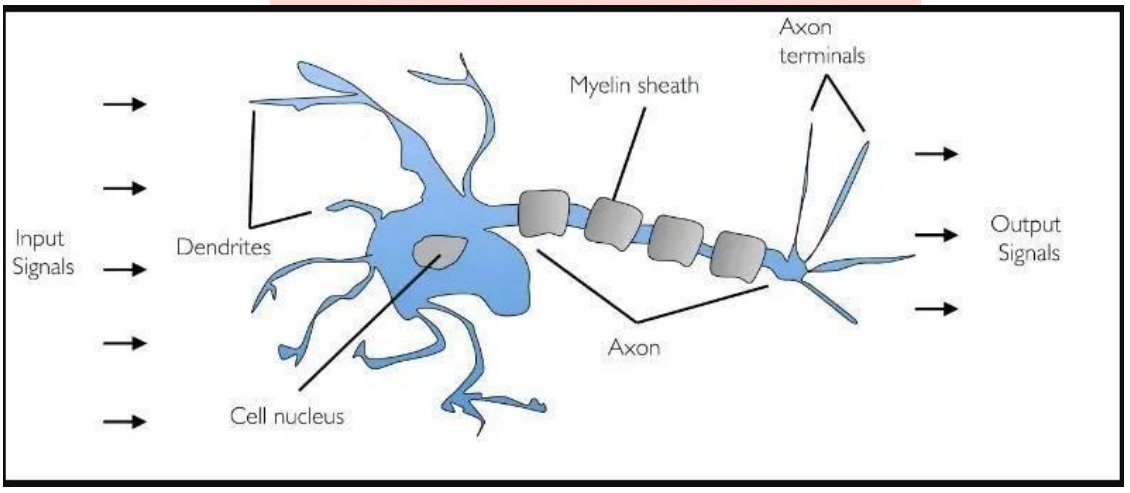
Outcome: At the end of this experiment, students will be able to demonstrate theperceptron learning law with its decision regions using python.

Hardware Requirement:

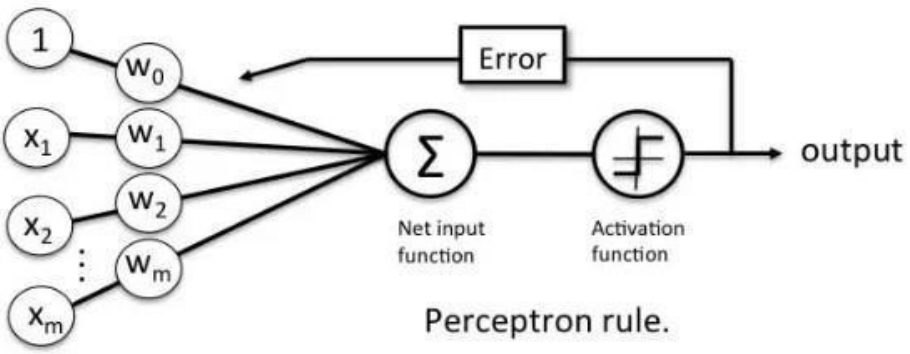
Software Requirement: Ubuntu OS,Python Editor(Python Interpreter)

Theory:

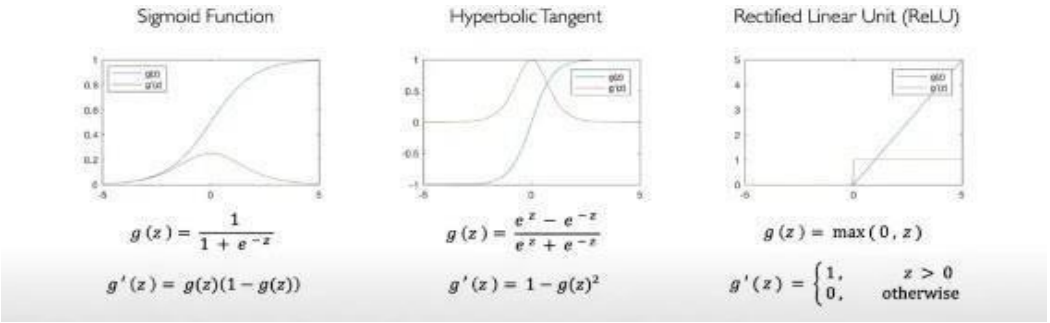
Perceptron is a machine learning algorithm which mimics how a neuron in the brain works.
It is also called as single layer neural network consisting of a single neuron. The output of this neural network is decided based on the outcome of just one activation function associated with the single neuron. In perceptron, the forward propagation of information happens. Deep neural network consists of one or more perceptrons laid out in two or more layers. Input to different perceptrons in a particular layer will be fed from previous layer by combining them with different weights.
Let's first understand how a neuron works. The diagram below represents a neuron in the brain. The input signals (x1, x2, …) of different strength (observed weights, w1, w2 …) is fed into the neuron cell as weighted sum via dendrites. The weighted sum is termed as the net input. The net input is processed by the neuron and output signal (observer signal in AXON) is appropriately fired. In case the combined signal strength is not appropriate based on decision function within neuron cell (observe activation function), the neuron does not fire any output signal.
The following is an another view of understanding an artificial neuron, a perceptron, in relation to a biological neuron from the viewpoint of how input and output signals flows:



The perceptron when represented as line diagram would look like the following with mathematical notations:



Step 1 – Input signals weighted and combined as net input: Weighted sumsof input signal reaches to the neuron cell through dendrites. The weighted inputs does represent the fact that different input signal may have different strength, and thus, weighted sum. This weighted sum can as well be termed as net input to the neuron cell.
Step 2 – Net input fed into activation function: Weighted The weighted sum of inputs or net input is fed as input to what is called as activation function. The activation function is a non-linear activation function. The activation functions are of different types such as the following: Unit step function, Sigmoid function, Rectilinear (ReLU) function, Hyperbolic tangent.
The diagram below depicts different types of non-linear activation functions.



Step 3A – Activation function outputs binary signal appropriately: The activation function processes the net input based on the unit step (Heaviside) function and outputs the binary signal appropriately as either 1 or 0. The activation function for perceptron can be said to be a unit step function. Recall that the unit step function, u(t), outputs the value of 1 when t >= 0 and 0 otherwise. In the case of a shifted unit step function, the function u(t-a) outputs the value of 1 when t >= a and 0 otherwise.
Step 3B – Learning input signal weights based on prediction vs actual: A parallel step is a neuron sending the feedback to strengthen the input signal strength (weights) appropriately such that it could create an output signal appropriately that matches the actual value. The feedback is based on the outcome of the activation function which is a unit step function. Weights are updated based on the gradient descent learning algorithm. Here is my post on gradient descent – Gradient descent explained simply with examples. Here is the equation based on which the weights get updated:

$$w_i \leftarrow w_i + \Delta w_i$$

where

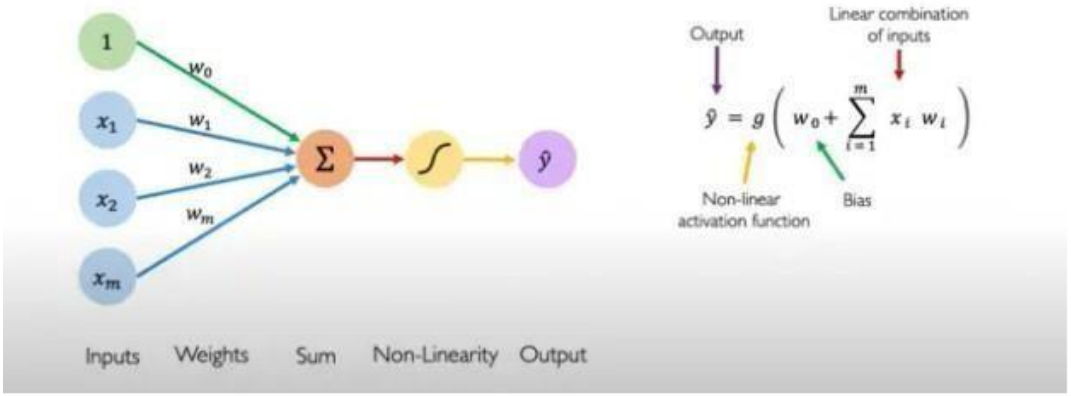
$$\Delta w_i = \eta(t - o)x_i$$

learning rate — target value — perceptron output — input value

- Weights get updated by δw, δw is derived by taking the first-order derivative of the loss function (gradient) and multiplying the output with negative (gradient descent) of learning rate. The output is what is shown in the above equation – the product of learning rate, the difference between actual and predicted value (perceptron output), and input value.
- The weights are updated based on each training example such that the perceptron can learn to predict closer to the actual output for the next input signal. This is also called stochastic gradient descent (SGD). Here is my post on stochastic gradient descent. That said, one could also try batch gradient descent to learn the weights of input signals.

Perceptron – A single-layer neural network comprising of a single neuron

Here is another picture of Perceptron that represents the concept explained above.



Conclusion: -

_____

_____

Questions:

Q1. What do you mean by Perceptron?
Q2. What are the different types of Perceptrons?
Q3. What is the use of the Loss functions?

EXPERIMENT NO. 5 (Group A)

Aim: Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation

Outcome: At end of this experiment, student will be able to Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation

Hardware Requirement:

Software Requirement: Ubuntu OS,Python Editor(Pyhton Interpreter)

Theory:

Backpropagation (short for "backward propagation of errors") is a popular algorithm used in artificial neural networks to train the weights of the network's connections. The goal of backpropagation is to adjust the weights of the network in order to minimize the difference between the predicted output and the actual output.

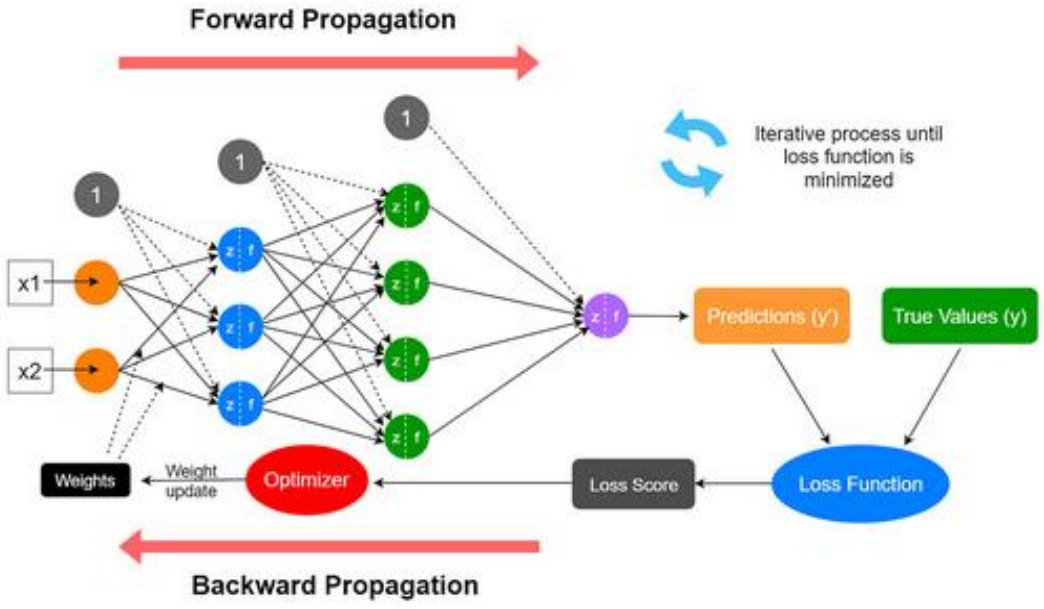
Backpropagation works by first making a forward pass through the network to obtain a prediction. The difference between the predicted output and the actual output is then calculated, and this difference (the error) is propagated backwards through the network. The algorithm then adjusts the weights of the connections in the network to reduce the error.

The adjustment of the weights is done using a process called gradient descent, which involves calculating the gradient (derivative) of the error with respect to each weight in the network. The weights are then updated by moving in the direction of the negative gradient, which helps to reduce the error.

The backpropagation algorithm is often used in conjunction with an optimization algorithm such as stochastic gradient descent (SGD) to efficiently update the weights of the network. It is a powerful technique that has been used to train neural networks for a wide range of applications, including image recognition, speech recognition, and natural language processing. A

Forward propagation, also known as feedforward, is the process of transmitting input data through a neural network in order to obtain an output prediction.

In a neural network, forward propagation involves a series of calculations that are performed on the input data as it passes through the network's layers. Each layer of the network consists of a set of neurons, which are connected to neurons in the previous layer by weighted connections.



The input data is fed into the first layer of the network, and each neuron in the layer calculates a weighted sum of its inputs, which is then passed through an activation function to produce an output value. This output is then passed to the next layer of the network as input, and the process is repeated until the output of the final layer is obtained.
The weights of the connections between the neurons are typically initialized randomly and then adjusted during the training process using an algorithm such as backpropagation. The goal of the training process is to adjust the weights so that the network's predictions become more accurate over time.
Forward propagation is a key component of neural network training and inference. It allows the network to process input data and produce output predictions, which can be compared to the actual output values in order to calculate an error. This error can then be used to update the network's weights and improve its accuracy.
The training process consists of the following steps:

Forward Propagation:
Take the inputs, multiply by the weights (just use random numbers as weights)
Let Y = WiIi = W1I1+W2I2+W3I3
Pass the result through a sigmoid formula to calculate the neuron's output. The Sigmoid function is used to normalize the result between 0 and 1:
1/1 + e-y

Back Propagation
Calculate the error i.e the difference between the actual output and the expected output. Depending on the error, adjust the weights by multiplying the error with the input and again with the gradient of the Sigmoid curve:
Weight += Error Input Output (1-Output) ,here Output (1-Output) is derivative of sigmoid curve.
Note: Repeat the whole process for a few thousand iterations.

Let's code up the whole process in Python. We'll be using the Numpy library to help us with all the calculations on matrices easily. You'd need to install a numpy library on your system to run the code
Command to install numpy: sudo apt -get install python-numpy

Conclusion: -

_____

_____

_____

Questions:

Q1. What Is Forward And Backward Propagation?
Q2. How do Forward And Backward Propagation work?
Q3. Write Difference between Forward And Backward Propagation?
Q4. What are steps involved in Forward Propagation?
Q5. What are steps involved in Backward Propagation?
Q6.What is Preactivation and activation in Forward Propagation?

EXPERIMENT NO. 6 (Group A)

Aim: Write a python Program for Bidirectional Associative Memory with two pairs of vectors.

Outcome: At end of this experiment, student will be able to study Bidirectional Associative Memory with two pairs of vectors in Python and use it to check output pattern.

Hardware Requirement:

Software Requirement: Ubuntu OS,Python Editor.

Theory:

Bidirectional associative memory (BAM), first proposed by Bart Kosko in the year 1988.
The BAM network performs forward and backward associative searches for stored stimulus responses. The BAM is a recurrent hetero associative pattern-marching network that encodes binary or bipolar patterns using Hebbian learning rule. It associates patterns, say from set A to patterns from set B and vice versa is also performed. BAM neural nets can respond to input from either layers (input layer and output layer).
The architecture of BAM network consists of two layers of neurons which are connected by directed weighted pare interconnections. The network dynamics involve two layers of interaction. The BAM network iterates by sending the signals back and forth between the two layers until all the neurons reach equilibrium. The weights associated with the network are bidirectional. Thus, BAM can respond to the inputs in either layer.

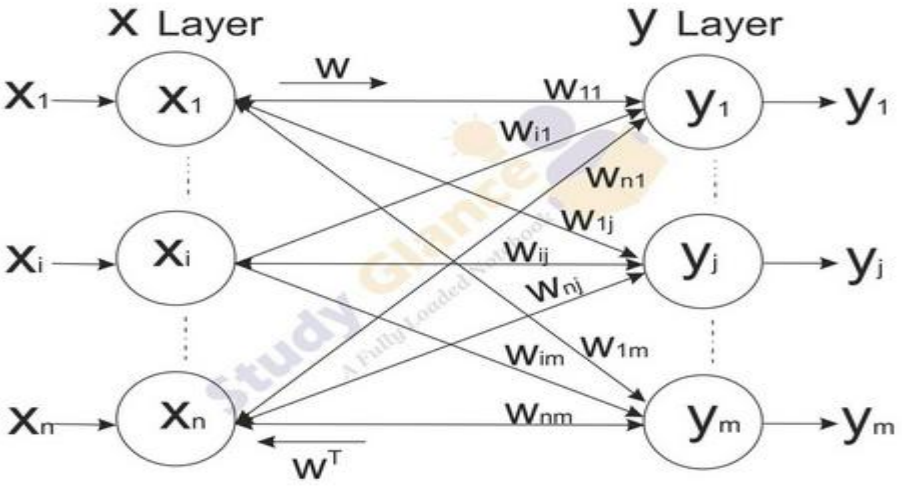


Figure shows a BAM network consisting of n units in X layer and m units in Y layer. The layers can be connected in both directions(bidirectional) with the result the weight matrix sent from the X layer to the Y layer is W and the weight matrix for signals sent from the Y layer to the X layer is W^T. Thus, the Weight matrix is calculated in both directions.

How does BAM Work?

1. Training: BAM is trained on paired patterns. Each pair consists of patterns from two sets (let's call them X and Y). The weight matrix is adjusted based on the outer product of these patterns.

2. Recall: Given an incomplete or noisy pattern from set X (or Y), BAM iteratively updates the neurons in both sets until the network stabilizes to a known pattern in set Y (or X).
3. Stability: One of the unique features of BAM is its guaranteed stability. It ensures that the recalled patterns won't oscillate indefinitely.

Algorithm :

- **Storage (Learning):** In this learning step of BAM, weight matrix is calculated between M pairs of patterns (fundamental memories) are stored in the synaptic weights of the network following the equation

$$W = \sum_{m=1}^{M} X_m Y_m^T$$

- **Testing**: We have to check that the BAM recalls perfectly Y_m for corresponding X_m and recalls X_m for corresponding Y_m. Using,

$$Y_m = \text{sign}\left(W^T X_m\right), \quad m = 1.2, \ldots, M$$

$$X_m = \text{sign}\left(W Y_m\right), \quad m = 1.2, \ldots, M$$

All pairs should be recalled accordingly.

- **Retrieval:** For an unknown vector *X* (a corrupted or incomplete version of a pattern from set *A* or *B*) to the BAM and retrieve a previously stored association: $X \neq X_m, \quad m = \mathbf{1}, \mathbf{2}, \ldots, M$

  - **Initialize the BAM:**
    $X(0) = X, \quad p = 0$
  - **Calculate the BAM output at iteration:** p
    $Y(p) = \text{sign}\left[W^T X(p)\right]$
  - **Update the input vector :** $X(p)$
    $X(p+1) = \text{sign}[WY(p)]$

- Repeat the iteration until convergence, when input and output remain unchanged.

Conclusion: -

_____

_____

_____

Questions:

1. Hetero-associative memory is also known as?
2. What is the objective of BAM?
3. A greater value of 'p' the vigilance parameter leads to?

EXPERIMENT NO. 7 (Group B)

Aim: Write a python program to show Back Propagation Network for XOR function with Binary Input and Output

Outcome: At end of this experiment, student will be able to to show Back Propagation Network for XOR function with Binary Input and Output

Hardware Requirement:

Software Requirement: Python IDE

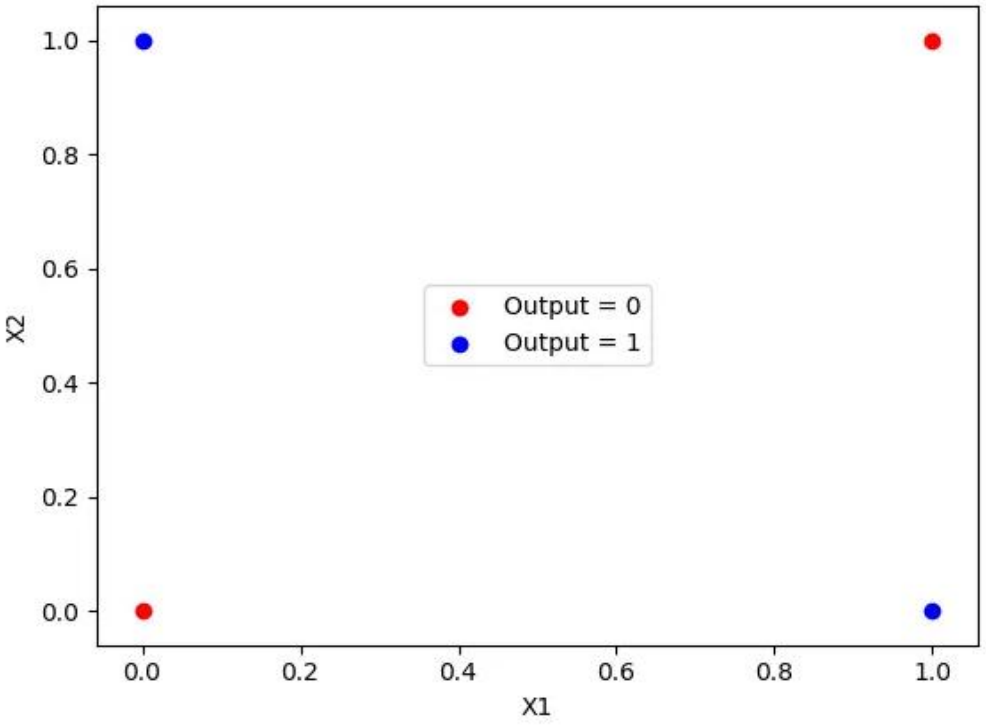
Theory:

Implementing logic gates using neural networks help understand the mathematical computation by which a neural network processes its inputs to arrive at a certain output. This neural network will deal with the XOR logic problem. An XOR (exclusive OR gate) is a digital logic gate that gives a true output only when both its inputs differ from each other. The truth table for an XOR gate is shown below:

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table for XOR

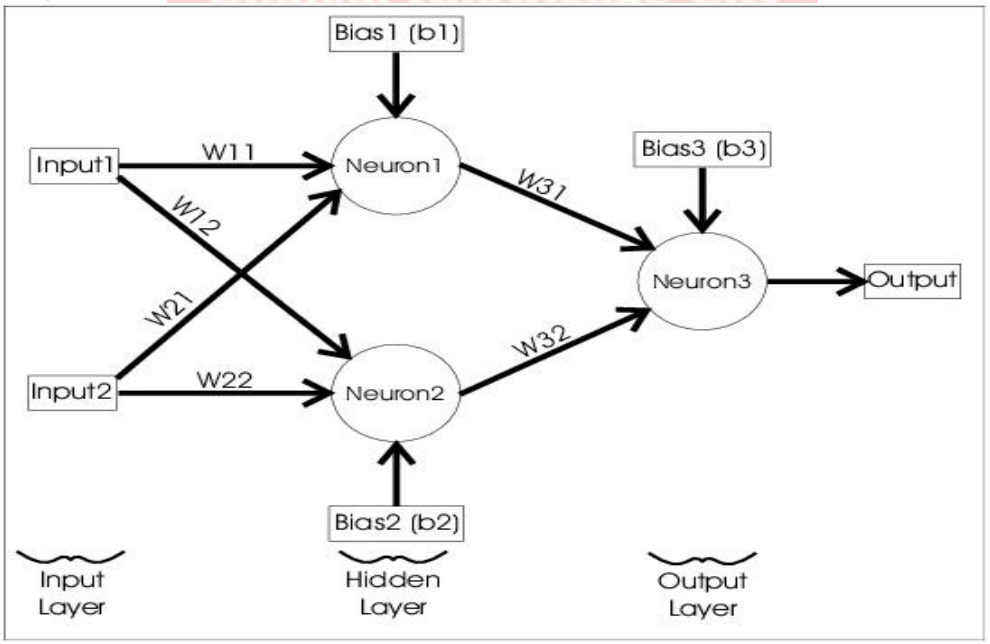
The goal of the neural network is to classify the input patterns according to the above truth table. If the input patterns are plotted according to their outputs, it is seen that these points are not linearly separable. Hence the neural network has to be modeled to separate these input patterns using decision planes.



XOR, Graphically

THE NEURAL NETWORK MODEL

As mentioned before, the neural network needs to produce two different decision planes to linearly separate the input data based on the output patterns. This is achieved by using the concept of hidden layers. The neural network will consist of one input layer with two nodes (X1,X2); one hidden layer with two nodes (since two decision planes are needed); and one output layer with one node (Y). Hence, the neural network looks like this:



The Neural Network Model to solve the XOR Logic (from: https://stopsmokingaids.me/)

THE SIGMOID NEURON

To implement an XOR gate, I will be using a Sigmoid Neuron as nodes in the neural network. The characteristics of a Sigmoid Neuron are:

1. Can accept real values as input.

2. The value of the activation is equal to the weighted sum of its inputs
i.e. ∑wi xi
3. The output of the sigmoid neuron is a function of the sigmoid function, which is also known as a logistic regression function. The sigmoid function is a continuous function which outputs values between 0 and 1.

THE LEARNING ALGORITHM

The information of a neural network is stored in the interconnections between the neurons i.e. the weights. A neural network learns by updating its weights according to a learning algorithm that helps it converge to the expected output. The learning algorithm is a principled way of changing the weights and biases based on the loss function.

Steps:

1. Initialize the weights and biases randomly.
2. Iterate over the data
   i. Compute the predicted output using the sigmoid function
   ii. Compute the loss using the square error loss function
   iii. W(new) = W(old) — α ΔW
   iv. B(new) = B(old) — α ΔB
3. Repeat until the error is minimal.

This is a fairly simple learning algorithm consisting of only arithmetic operations to update the weights and biases. The algorithm can be divided into two parts: the forward pass and the backward pass also known as "backpropagation."

Inputs

$$x_1 = [1,1]^T, \quad y_1 = +1$$
$$x_2 = [0,0]^T, \quad y_2 = +1$$
$$x_3 = [1,0]^T, \quad y_3 = -1$$
$$x_4 = [0,1]^T, \quad y_4 = -1$$

2 hidden neurons are used, each takes two inputs with different weights. After each forward pass, the error is back propagated. Here sigmoid is used as the activation function at the hidden layer.

At hidden layer:

$$H_1 = x_1 w_1 + x_2 w_2$$
$$H_2 = x_1 w_3 + x_2 w_4$$

At output layer:

$$\hat{Y} = \sigma(H_1) w_5 + \sigma(H_2) w_6$$

here, σ represents sigmoid function.

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

Loss function:

$$\frac{1}{2}(Y - \hat{Y})^2$$

Conclusion: -

_____

_____

_____

Questions:

Q1. What is the backpropagation algorithm for XOR gate?
Q2. How do you solve XOR with a neural network?
Q3. What is the implementation of artificial neural network for XOR?

EXPERIMENT NO. 8 (Group B)

Aim: Write a python program to design a Hopfield Network which stores 4 vectors.

Outcome: At end of this experiment, student will be able to design a Hopfield Network.

Hardware Requirement:

Software Requirement: Python IDE

Theory:

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of *'n'* fully connected recurrent neurons. It is generally used in performing auto association and optimization tasks. It is calculated using a converging interactive process and it generates a different response than our normal neural nets.

Hopfield network is a special kind of neural network whose response is different from other neural networks. It is calculated by converging iterative process. It has just one layer of neurons relating to the size of the input and output, which must be the same. When such a network recognizes, for example, digits, we present a list of correctly rendered digits to the network. Subsequently, the network can transform a noise input to the relating perfect output.

Discrete Hopfield Network: It is a fully interconnected neural network where each unit is connected to every other unit. It behaves in a discrete manner, i.e. it gives finite distinct output, generally of two types:

- **Binary (0/1)**
- **Bipolar (-1/1)**

The weights associated with this network is symmetric in nature and has thefollowing properties.

Structure & Architecture

- Each neuron has an inverting and a non-inverting output.
- Being fully connected, the output of each neuron is an input to all other neurons but not self.

Fig 1 shows a sample representation of a Discrete Hopfield Neural Network architecture having the following elements.

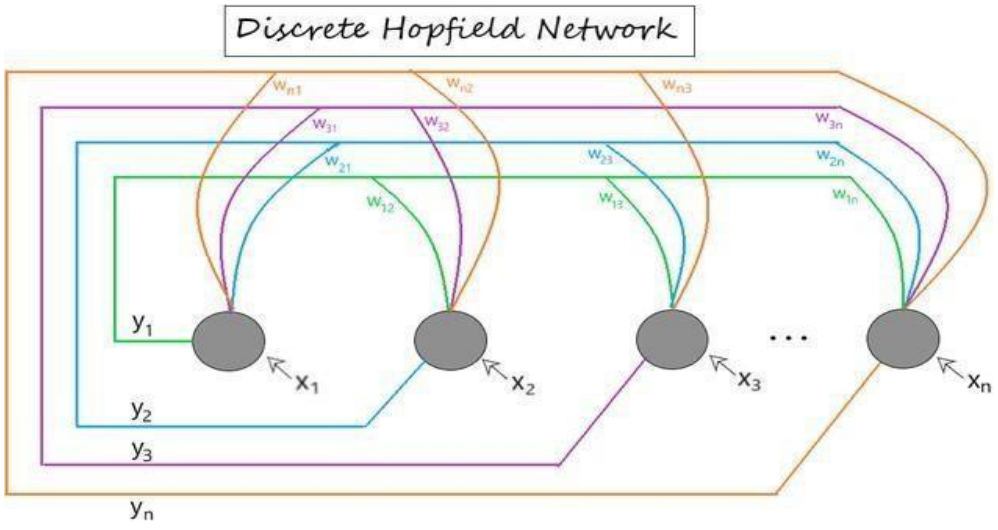


Fig 1: Discrete Hopfield Network Architecture

Training Algorithm

For storing a set of input patterns S(p) [p = 1 to P], where S(p) = S1(p) ... Si(p) ... Sn(p), the weight matrix is given by:

For binary patterns

$w_{ij} = \sum_{p = 1}^{P} [2s_{i}(p) - 1][2s_{j}(p) - 1]\ (w_{ij}\ for\ all\ i\neq j)$

For bipolar patterns

$w_{ij} = \sum_{p = 1}^{P} [s_{i}(p)s_{j}(p)]\ (where\ w_{ij} = 0\ for\ all\ i=j)$

(i.e. weights here have no self connection)

Steps Involved

Step 1 - Initialize weights (wij) to store patterns (using trainingalgorithm).

Step 2 - For each input vector yi, perform steps 3-7.

Step 3 - Make initial activators of the network equal to the external input vector x.

$y_i = x_i : (for\ i = 1\ to\ n)$

Step 4 - For each vector yi, perform steps 5-7.

Step 5 - Calculate the total input of the network yin using the equation given below.

$y_{in_{i}} = x_i + \sum_{j} [y_jw_{ji}]$

Step 6 - Apply activation over the total input to calculate the output as per the equation given below:

$y_i = \begin{cases} & 1 \text{ if } y_{in}>\theta_i \\ &y_i \text{ if } y_{in}=\theta_i \\ & 0 \text{ if } y_{in}<\theta_i \end{cases}$

(where θi (threshold) and is normally taken as 0)

Step 7 - Now feedback the obtained output yi to all other units. Thus, the activation vectors are updated.

Step 8 - Test the network for convergence.

Example:

Suppose we have only two neurons: N = 2

There are two non-trivial choices for connectivities:

w12 = w21 = 1

w12= w21 = -1

Asynchronous updating:

In the first case, there are two attracting fixed points termed as [-1,-1] and [-1,-1]. All orbit converges to one of these. For a second, the fixed points are [-1,1] and [1,-1], and all orbits are joined through one of these. For any fixed point, swapping all the signs gives another fixed point.

Synchronous updating:

In the first and second cases, although there are fixed points, none can be attracted to nearby points, i.e., they are not attracting fixed points. Some orbits oscillate forever.

Energy function evaluation:

Hopfield networks have an energy function that diminishes or is unchanged with asynchronous updating.

For a given state $X \in \{-1, 1\}$ N of the network and for any set of association weights Wij with Wij = wji and wii =0 let,

Hopfield Network

Here, we need to update Xm to X'm and denote the new energy by E' and show that.

E'-E = (Xm-X'm) ∑i≠mWmiXi.

Using the above equation, if Xm = Xm' then we have E' = E

If Xm = -1 and Xm' = 1 , then Xm - Xm' = 2 and hm= ∑iWmiXi ? 0

Thus, E' - E ≤ 0

Similarly if Xm =1 and Xm'= -1 then Xm - Xm' = 2 and hm= ∑iWmiXi< 0

Thus, E - E' < 0.

Conclusion: -

_____

_____

_____

Questions:

Q1. Write note on competitive learning ?
Q2. How SOM works?
Q3. What are the issues faced while training in Recurrent Networks?
Q4. Explain the different layers of CNN?
Q5. Explain Hopfield Model?

EXPERIMENT NO. 9 (Group C)

Aim: How to Train a Neural Network with TensorFlow/Pytorch and evaluation of logistic regression using tensorflow.

Outcome: At end of this experiment, student will be able to Train a Neural Networkwith TensorFlow/Pytorch and evaluation of logistic regression using tensorflow.

Hardware Requirement:

Software Requirement: Python IDE, Spyder, Tensorflow.

Theory:

What is regression?

For example, if the model that we built should predict discrete or continuous values like a person's age, earnings, years of experience, or need to find out that how these values are correlated with the person, it shows that we are facing a regression problem.

What is a neural network?

Just like a human brain, a neural network is a series of algorithms that detect basic patterns ina set of data. The neural network works as a neural network in the human brain. A "neuron" in a neural network is a mathematical function that searches for and classifies patternsaccording to a specific architecture.

It is possible and important to talk about each of these topics in detail and for a long time, butmy goal in this article is *to build a model* and work on it together after briefly touching on theimportant points. If you start to write codes with me and get the results by yourself, everything will be more fun and memorable. *So let's get our hands dirty.*

First, let's start with **importing some libraries** that we will use at the beginning:

```
import tensorflow as tf print(tf. version_)
import numpy as np
import matplotlib.pyplotas plt
```

What is regression?

For example, if the model that we built should predict discrete or continuous values like a person's age, earnings, years of experience, or need to find out that how these values are correlated with the person, it shows that we are facing a regression problem.

What is a neural network?

Just like a human brain, a neural network is a series of algorithms that detect basic patterns in a set of data. The neural network works as a neural network in the human brain. A "neuron" in a neural network is a mathematical function that searches for and classifies patterns according to a specific architecture.

Transform into an expert and significantly impact the world of data science.

First, let's start with **importing some libraries** that we will use atthe beginning:

```
import tensorflow as tf print(tf. version_)
import numpy as np importmatplotlib.pyplotas plt
```

We are dealing with a regression problem, and we will **create our dataset:**

One important point in NN is the input shapes and the output shapes. The input shape is the shape of the data that we train the model on, and the output shape is the shape of data that we expect to come out of our model. Here we will use X and aim to predict y, so, X is our input and y is our output.

```
X. shape,y.shape
>>((74,),(74,))
```

Let's start building our model with TensorFlow. There are 3 typical steps to creating a model in TensorFlow:

- **Creating a model** – connect the layers of the neural network yourself, here we either use Sequential or Functional API, also we may import a previously built model that we call transfer learning.
- **Compiling a model** – at this step, we define how to measure a model's performance, which optimizer should be used.
- **Fitting a model** – In this step, we introduce the model to the data and let it find patterns.

We've created our dataset, that is why we can directly start modeling, but first, we need to split our train and test set.

```
len(X)
>> 74
X_train=X[:60]y_train
=y[:60]
X_test=X[60:]y_test
=y[60:]
len(X_train),len(X_test)
>> (60,14)
```

The best way of getting more insight into our data is by **visualizing** it! So, let's do it!

```
plt.figure(figsize=(12,6))
plt.scatter(X_train, y_train,c='b',label='Trainingdata')plt.scatter(X_test, y_test, c='g',
label='Testing data') plt.legend()
```


Improve the Regression model with neural network

```
tf.random.set_seed(42)
model_2
=tf.keras.Sequential([tf.keras.l
ayers.Dense(1),
tf.keras.layers.Dense(1)])
model_2.compile(loss=tf.keras.losses.mae,
optimizer=tf.keras.optimizers.SGD(), metrics=['mae'])
model_2.fit(X_train,y_train,epochs=100,verbose=0)
```

Here we just replicated the first model, and add an extra layer to see how it works?

```
preds_2 =model_2.predict(X_test)
plot_preds(predictions=preds_2)
```

It allows categorizing data into discrete classes by learning the relationship from a given set of labeled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function.

In case of Logistic regression, the hypothesis is the Sigmoid of a straight line,i.e,h(x) = \sigma(wx + b) where \sigma(z) = \frac{1}{1 + e^{-z}}

Where the vector w represents the Weights and the scalar b represents the Bias of the model.

Conclusion: -

_____

_____

Questions:

Q1. Write short note on Softmax regression?
Q2. What are the deep learning frameworks or tools?
Q3. What are the applications of deep learning?
Q4. What is the meaning of term weight initialization in neural networks?
Q5. What are the essential elements of PyTorch?

EXPERIMENT NO. 10 (Group C)

Aim: TensorFlow/Pytorch implementation of CNN.
Outcome: At end of this experiment, student will be able to implementTensorFlow/ Pytorch of CNN

Hardware Requirement:

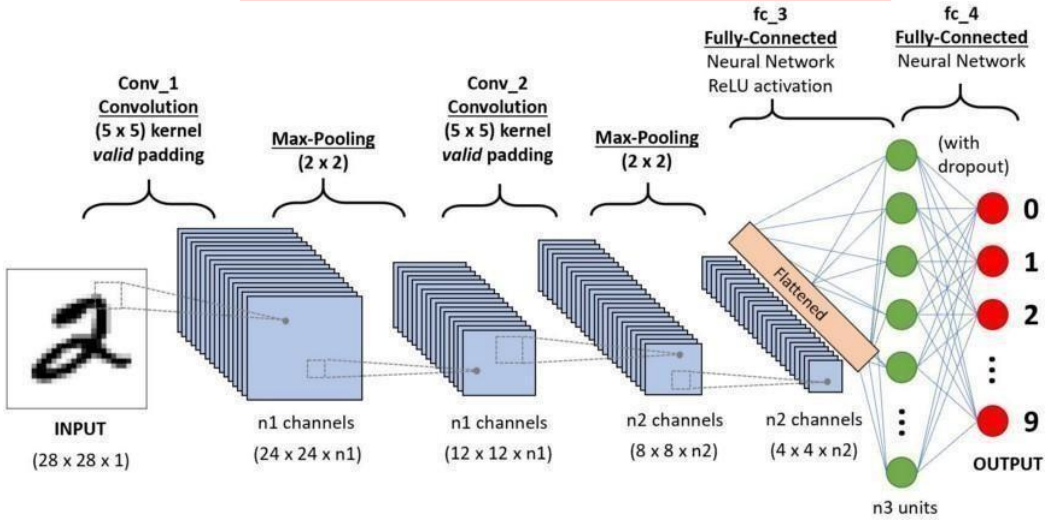
Software Requirement: Python IDE, Spyder, Tensorflow.

Theory:

Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by YannLeCun, a computer science researcher who worked in the background. LeCun built on the work of Kunihiko Fukushima, a Japanese scientist, a basic network for image recognition.

The old version of CNN, called LeNet (after LeCun), can see handwritten digits. CNN helps find pin codes from postal. But despite their expertise, ConvNets stayed close to computer vision and artificial intelligence because they faced a major problem: They could not scale much. CNN's require a lot of data and integrate resources to work well for large images.

At the time, this method was only applicable to low-resolution images. Pytorch is a library that can do deep learning operations. We can use this to perform Convolutional neural networks. Convolutional neural networks contain many layers of artificial neurons. Synthetic neurons, complex simulations of biological counterparts, are mathematical functions that calculate the weighted mass of multiple inputs and product value activation.



The above image shows us a CNN model that takes in a digit-like image of 2 and gives us the result of what digit was shown in the image as a number. We will discuss in detail how we getthis in this article.

CIFAR-10 is a dataset that has a collection of images of 10 different classes. This dataset is widely used for research purposes to test different machine learning models and especially for computer vision problems. In this article, we will try to build a Neural network model using Pytorch and test it on the CIFAR-10 dataset to check what accuracy of prediction can be obtained.

Importing the PyTorch Library

```
importnumpy as np
import pandasas pd
import torch
importtorch.nn.functional as F fromtorchvisionimport
datasets,transformsfrom torch import nn
importmatplotlib.pyplot as plt importnumpy as np
importseabornassns
#fromtqdm.notebookimporttqdm
fromtqdm import tqdm
```

In this step, we import the required libraries. We can see we use NumPy for numerical operations and pandas for data frame operations. The torch library is used to import Pytorch.

Pytorch has an nn component that is used for the abstraction of machine learning operations and functions. This is imported as F. The torchvision library is used so that we can import theCIFAR-10 dataset. This library has many image datasets and is widely used for research. The transforms can be imported so that we can resize the image to equal size for all the images. The tqdm is used so that we can keep track of the progress during training and is used for visualization.

Analyzing the data with PyTorch

```
print("Number of points:",trainData.shape[0]) print("Numberoffeatures:", trainData.shape[1])
print("Features:",trainData.columns.values) print("Number of Unique Values")
for col in trainData:
print(col,":",len(trainData[col].unique()))
plt.figure(figsize=(12,8))
```

Output:

```
Number of points: 50000
Number of features: 2 Features:
['id''label']Number of Unique
Valuesid : 50000
label : 10
```

In this step, we analyze the dataset and see that our train data has around 50000 rows withtheir id and associated label. There is a total of 10 classes as in the name CIFAR-10.

Getting the validation set using PyTorch

```
fromtorch.utils.dataimportrandom_splitval_size = 5000
train_size=len(dataset)- val_size
train_ds,val_ds=random_split(dataset,[train_size,val_size])len(train_ds), len(val_ds)
```

This step is the same as the training step, but we want to split the data into train andvalidation sets.

```
(45000,5000)
fromtorch.utils.data.dataloaderimportDataLoader
batch_size=64
train_dl=DataLoader(train_ds,batch_size, shuffle=True, num_workers=4,pin_memory=True)
val_dl=DataLoader(val_ds,batch_size,num_workers=4,pin_memory=True)
```

The torch.utils have a data loader that can help us load the required data bypassing variousparams like worker number or batch size.

Defining the required functions

```
@torch.no_grad()
defaccuracy(outputs,labels):
    _, preds = torch.max(outputs, dim=1) returntorch.tensor(torch.sum(preds==labels).item()/len(preds))

classImageClassificationBase(nn.Module):
deftraining_step(self, batch):
images, labels = batch
out=self(images)                    #Generatepredictionsloss =
F.cross_entropy(out, labels)   # Calculate loss
accu=accuracy(out,labels)
returnloss,accu

defvalidation_step(self,batch):
images, labels = batch
out=self(images)                                    #Generatepredictionsloss=
F.cross_entropy(out,labels)                         # Calculate loss
acc=accuracy(out, labels)                           #Calculateaccuracyreturn
{'Loss': loss.detach(), 'Accuracy': acc}

defvalidation_epoch_end(self,outputs):
batch_losses =[x['Loss'] forxinoutputs]
epoch_loss=torch.stack(batch_losses).mean()                    #Combinelosses
batch_accs = [x['Accuracy'] for x in outputs]
epoch_acc=torch.stack(batch_accs).mean()                       #Combineaccuraciesreturn
{'Loss': epoch_loss.item(), 'Accuracy':epoch_acc.item()}

defepoch_end(self,epoch,result):
print("Epoch:",epoch+ 1)
        print(f'Train Accuracy:{result["train_accuracy"]*100:.2f}% Validation
Accuracy:{result["Accuracy"]*100:.2f}%')
print(f'TrainLoss:{result["train_loss"]:.4f} ValidationLoss:{result["Loss"]:.4f}')
```

Convolutional neural networks, also called ConvNets, were first introduced in the 1980s by YannLeCun, a computer science researcher who worked in the background. LeCun built on the work of Kunihiko Fukushima, a Japanese scientist, a basic network for image recognition.

The old version of CNN, called LeNet (after LeCun), can see handwritten digits. CNN helpsfind pin codes from postal. But despite their expertise, ConvNets stayed close to computer vision and artificial intelligence because they faced a major problem: They could not scale much. CNN's require a lot of data and integrate resources to work well for large images.

At the time, this method was only applicable to low-resolution images. Pytorch is a library that can do deep learning operations. We can use this to perform Convolutional neural networks. Convolutional neural networks contain many layers of artificial

neurons. Synthetic neurons, complex simulations of biological counterparts, are mathematical functions that
calculate the weighted mass of multiple inputs and product value activation.

As we can see here we have used class implementation of Image Classification and it takes one parameter that is nn.Module. Within this class, we can implement the various functions or various steps like training, validation, etc. The functions here are simple python implementations.

Questions:

Q1. How do we find the derivatives of the function in PyTorch?
Q2. Give any one difference between torch.nn and torch.nn.functional?
Q3. Why it is difficult for the network is showing the problem?
Q4. Are tensor and matrix the same?
Q5. What is PyTorch?

EXPERIMENT NO. 11 (Group C)

Aim: MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow.

Outcome: At end of this experiment, student will be able to detect Handwritten Character using PyTorch, Keras and Tensorflow.

Hardware Requirement:

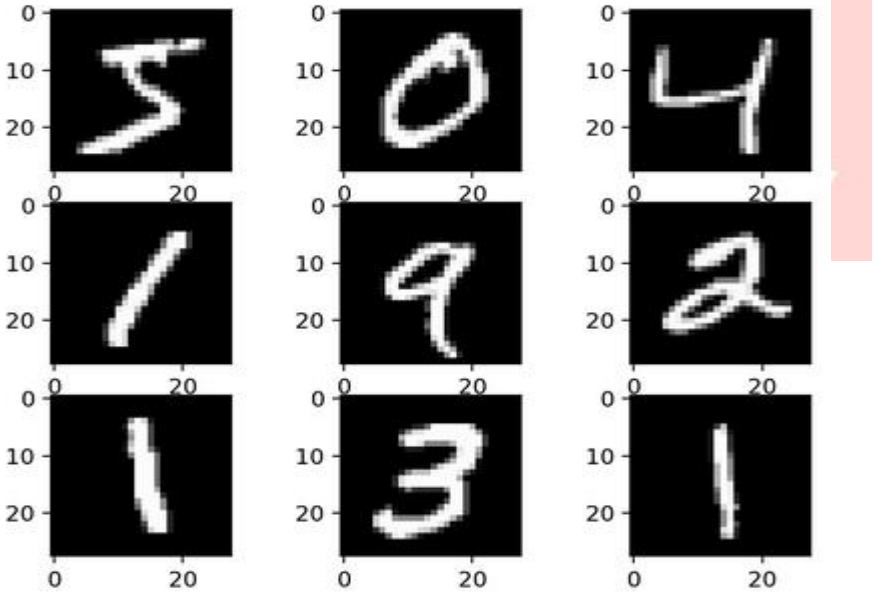
Software Requirement: Python IDE

Theory:

The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning.

Although the dataset is effectively solved, it can be used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch. This includes how to develop a robusttest harness for estimating the performance of the model, how to explore improvements to the model, and how to save the model and later load it to make predictions on new data.

MNIST Handwritten Digit Classification Dataset:



The MNIST dataset is an acronym that stands for the Modified NationalInstitute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwrittensingle digits between 0 and 9. The task is to classify a given image of a handwritten digit into one of 10 classesrepresenting integer values from 0 to 9, inclusively. It is a widely used and deeply understood dataset and, for the most part, is "*solved*." Top-performing models are deep learning convolutional neural networks that achievea classification accuracy of above 99%, with an error rate between 0.4 %and 0.2% on the holdout test dataset. The example below loads the MNIST dataset using the Keras API and creates a plot ofthe first nine images in the training dataset.

example of loading the mnist dataset from tensorflow.keras.datasets importmnistfrom matplotlib import pyplot as plt

```
# load dataset
(trainX, trainy), (testX, testy) = mnist.load_data()     # summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape,testy.shape))  # plot first fewimagesfor i
in range(9):
# define subplot
plt.subplot(330 + 1 +i)    # plot raw pixel data
plt.imshow(trainX[i],
cmap=plt.get_cmap('gray'))                # show the figureplt.show()
```

Running the example loads the MNIST train and test dataset and prints their shape. We can see that there are 60,000 examples in the training dataset and 10,000 in the testdataset and that images are indeed square with 28×28 pixels.

```
Train: X=(60000, 28, 28), y=(60000,)
Test: X=(10000, 28, 28), y=(10000,)
```

Model Evaluation Methodology

Although the MNIST dataset is effectively solved, it can be a useful starting point for developing and practicing a methodology for solving image classification tasks using convolutional neural networks.

Instead of reviewing the literature on well-performing models on the dataset, we can develop a new model from scratch.

The dataset already has a well-defined train and test dataset that we can use.

In order to estimate the performance of a model for a given training run, we can further split the training set into a train and validation dataset. Performance on the train and validation dataset over each run can then be plotted to provide learning curves and insight into how well a model is learning the problem.

The Keras API supports this by specifying the "*validation_data*" argument to the *model.fit()*function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

```
# record model performance on a validation dataset duringtraininghistory =
model.fit(..., validation_data=(valX, valY))
```

In order to estimate the performance of a model on the problem in general, we can use k-foldcross-validation, perhaps five-fold cross-validation. This will give some account ofthe models variance with both respect to differences in the training and test datasets, and in termsof the stochastic nature of the learning algorithm. The performance of a model can be taken as the mean performance across k-folds, given the standard deviation, that could be used to estimate a confidence interval if desired. We can use the KFold class from the scikit-learn API to implement the k-fold cross-validation evaluation of a given neural network model. There are many ways toachieve this, although we can choose a flexible approach where the *KFold* class is onlyused to specify the row indexes used for each spit.

```
# example of k-fold cv for a neural
netdata = …

# prepare cross validation
kfold = KFold(5, shuffle=True,
random_state=1)# enumerate splits
for train_ix, test_ix in
      kfold.split(data):model = ...
```

Conclusion: -

_____

_____

_____

_____

Questions:

Q1. What is Deep Learning?
Q2. Explain Difference between keras and tensorflow?
Q3. What is Convolution operation? Explain in details?
Q4. What is the simple working of an algorithm in TensorFlow?
Q5. What are the languages that are supported in TensorFlow?