Empowerment Through Quality Technical Education

AJEENKYA DY PATIL SCHOOL OF ENGINEERING

Dr. D. Y. Patil Knowledge City, Charholi (Bk), Lohegaon, Pune – 412 105

Website: https://dypsoe.in/

LAB MANUAL

Artificial Neural Network

(317531)

BE (AI&DS) 2019 COURSE

Course Coordinator

Prof. Payal Deshmukh

Prof. Priyanka Bhore

Prof. Gauri Thite

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Department of Artificial Intelligence & Data Science

Vision:

Imparting quality education in the field of Artificial Intelligence and Data Science

Mission:

- To include the culture of R and D to meet the future challenges in AI and DS.
- To develop technical skills among students for building intelligent systems to solve problems.
- To develop entrepreneurship skills in various areas among the students.
- To include moral, social and ethical values to make students best citizens of country.

Program Educational Outcomes:

1. To prepare globally competent graduates having strong fundamentals, domain knowledge, updated with modern technology to provide the effective solutions for engineering problems.
2. To prepare the graduates to work as a committed professional with strong professional ethics and values, sense of responsibilities, understanding of legal, safety, health, societal, cultural and environmental issues.
3. To prepare committed and motivated graduates with research attitude, lifelong learning, investigative approach, and multidisciplinary thinking.
4. To prepare the graduates with strong managerial and communication skills to work effectively as individuals as well as in teams.

Program Specific Outcomes:

1. Professional Skills- The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, networking, artificial intelligence and data science for efficient design of computer-based systems of varying complexities.

2. Problem-Solving Skills- The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.

3. Successful Career and Entrepreneurship- The ability to employ modern computer languages, environments and platforms in creating innovative career paths to be an entrepreneur and to have a zest for higher studies.

Table of Contents

Contents

1. Guidelines to manual usage

This manual assumes that the facilitators are aware of collaborative learning methodologies.

This manual will provide a tool to facilitate the session on Digital Communication modules in collaborative learning environment.

The facilitator is expected to refer this manual before the session.

Icon of Graduate Attributes

K Applying Knowledge	**A** Problem Analysis	**D** Design & Development	**I** Investigation of problems
M Modern Tool Usage	**E** Engineer & Society	**E** Environment Sustainability	**T** Ethics
T Individual & Team work	**O** Communication	**M** Project Management & Finance	**I** Life-Long Learning

Disk Approach- Digital Blooms Taxonomy

1: Remembering / Knowledge
2: Comprehension / Understanding
3: Applying
4: Analyzing
5: Evaluating
6: Creating / Design

Program Outcomes:

1. **Engineering knowledge:** An ability to apply knowledge of mathematics, including discrete mathematics, statistics, science, computer science and engineering fundamentals to model the software application.
2. **Problem analysis:** An ability to design and conduct an experiment as well as interpret data, analyze complex algorithms, to produce meaningful conclusions and recommendations.
3. **Design/development of solutions**: An ability to design and development of software system, component, or process to meet desired needs, within realistic constraints such as economic, environmental, social, political, health & safety, manufacturability, and sustainability.
4. **Conduct investigations of complex problems**:An ability to use research based knowledge including analysis, design and development of algorithms for the solution of complex problems interpretation of data and synthesis of information to provide valid conclusion.
5. **Modern tool usage:** An ability to adapt current technologies and use modern IT tools, to design, formulate, implement and evaluate computer based system, process, by considering the computing needs, limits and constraints.
6. **The engineer and society:** An ability of reasoning about the contextual knowledge of the societal, health, safety, legal and cultural issues, consequent responsibilities relevant to IT practices.
7. **Environment and sustainability:** An ability to understand the impact of engineering solutions in a societal context and demonstrate knowledge of and the need for sustainable development.
8. **Ethics:** An ability to understand and commit to professional ethics and responsibilities and norms of IT practice.
9. **Individual and team work :**An ability to apply managerial skills by working effectively as an individual, as a member of a team, or as a leader of a team in multidisciplinary projects.
10. **Communication:** An ability to communicate effectively technical information in speech, presentation, and in written form
11. **Project management and finance:** An ability to apply the knowledge of Information Technology and management principles and techniques to estimate time and resources needed to complete engineering project.
12. **Life-long learning:** An ability to recognize the need for, and have the ability to engage in independent and life-long learning.

Course Name:Artificial Neural Network

Course Code: (317531)

Course Outcomes

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

CO to PO Mapping:

PO/CO	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

CO to PSO Mapping:

	PSO1	PSO2	PSO3
CO1	2	2	-
CO2	2	2	-
CO3	2	2	-
CO4	2	2	1

2. Laboratory Objective

- Students will gain hands-on experience in understanding the basics of ANN models and pattern recognition
- Implement solutions that can classify, categorize and predict, based on available/provided information
- Design solutions for object identification, image recognition, speech recognition through pattern recognition, text classification and categorization.

3. Laboratory Equipment/Software

Software Requirements:

Jupyter Notebook / PyCharm

Hardware Requirements:
8/16 GB RAM
GPU (recommended)

4. Laboratory Experiment list

Sr. No	Title
	List of 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 recognize 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

4.1. Experiment No. 1

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

Objective: To learn about activation functions and perform its code in python.

Theory:

What are Activation Functions?

A neural network activation function is a function that introduces nonlinearity into the model. A neural network has multiple nodes in each layer, and in a fully connected network, every node in one layer is connected to every node in the next layer.

Why use activation functions?

1. Activation functions' main objective is to add non-linearity into the network so that it can model more intricate and varied interactions between inputs and outputs. In the absence of activation functions, the network would only be capable of performing linear transformations, which cannot adequately represent the complexity and nuances of real-world data. Since neural networks need to implement complex mapping functions, non-linear activation functions must be used to introduce the much-needed nonlinearity property that allows approximating any function.
2. Normalizing each neuron in the network's output is a key benefit of utilizing activation functions. Depending on the inputs it gets and the weights associated to those inputs, a neuron's output can range from extremely high to extremely low. Activation functions make ensuring that each neuron's output falls inside a defined range, which makes it simpler to optimise the network during training.

Types of Activation Function:

Linear Activation Function: The linear activation function, also known as "no activation," or "identity function" (multiplied x 1.0), is where the activation is proportional to the input. The function doesn't do anything to the weighted sum of the input, it simply spits out the value as it is

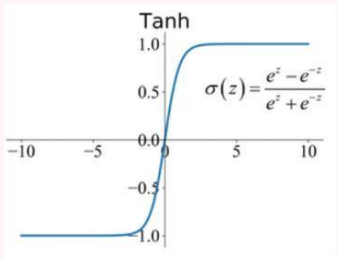
Linear f(x) =x

Sigmoid Activation Function: 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,

Sigmoid / Logistic f(x)= $1/(1 + e^{-x})$

Tanh Activation Function: 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.

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



Softmax Activation Function: Softmax functions are often written as a combination of multiple sigmoid. We know that Sigmoid returns a value between 0 and 1. This can be treated as the probability of a data point belonging to a particular class. Therefore, sigmoid are often used for binary classification problems. The softmax function can be used for multiclass classification problems. This function returns the probability of a data point belonging to each unique class. Here is the formula for the same –

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

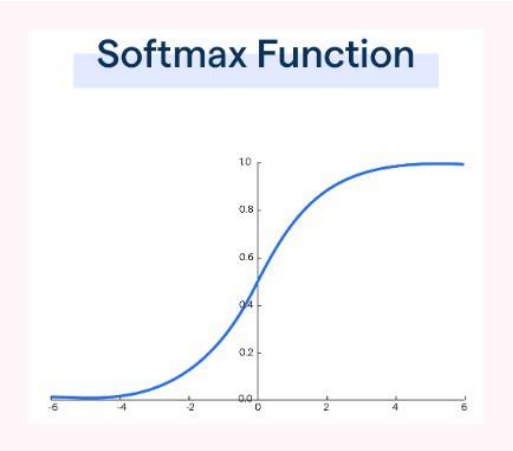
$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

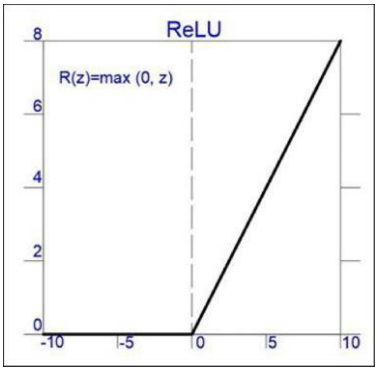
K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

e^{z_j} = standard exponential function for output vector



ReLU Function: ReLU stands for Rectified Linear Unit. 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.



$$\mathbf{f(x) = max(0, x)}$$

Applications:

1. Activation function decides whether a neuron should be activated or not.
2. The role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).

Input: Array

e.g. np.array([-1, 0, 1])

Output: Activation Function

```
print(sigmoid(x)) # [0.26894142 0.5 0.73105858]
print(tanh(x)) # [-0.76159416 0. 0.76159416]
print(relu(x)) # [0 0 1]
print(softmax(x)) # [0.09003057 0.24472847 0.66524096]
```

Conclusion:

We have successfully implemented program to plot a few activation functions that are being used in neural networks.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform activation function on element-wise operations on arrays.

Questions:

1. What is Activation Function?
2. What are the three activation functions?
3. Why do we use the ReLU activation function**?**
4. What are activation functions and their examples?
5. What is the use of Activation Function?

4.2. Experiment No. 2

Aim: Generate ANDNOT function using McCulloch-Pitts neural net.

Objective: Learning to design MP models for logic functions like ANDNOT

Theory:

McCulloch-Pitts neural model: 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 a function can be described mathematically using the equations

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.

Activation Function:

Output Function ANDNOT Function:

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

TRUTH TABLE:

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

ANN 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 0 = 1. Yin is an activation value
X1= 1, X2= 1,
Yin = W1*X1+W2*X2=1*1+(-1)*1=0, Yin< © , so Y=0
X1 = 1, X2 = 0
Yin = 1*1+0*(-1) = 1, Yin = ©, so Y = 1
X1 = 0, X = 1
Yin = 0-1 = -1, Yin< 8, so Y = 0
X1 = 0, X2 = 0
Yin = 0, Yin< 8, so Y = 0
So, Y = [010 0]

Applications:

1. To design logical operations
2. To implement basic logic gates like AND, OR, and NOT

Input:

Weights of Neuron:
w1=1
w2=-1
Threshold:
Theta=1

Output:

w1=1
w2=-1
Threshold:
Theta=1
With Output of Neuron:
0100

Conclusion:

We have successfully implemented ANDNOT function using McCulloch-Pitts neural net.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform basic logic gates like AND, OR, and NOT.

Questions:

1. What are the limitations of the McCulloch-Pitts neuron model?
2. What is the MP neural network model?

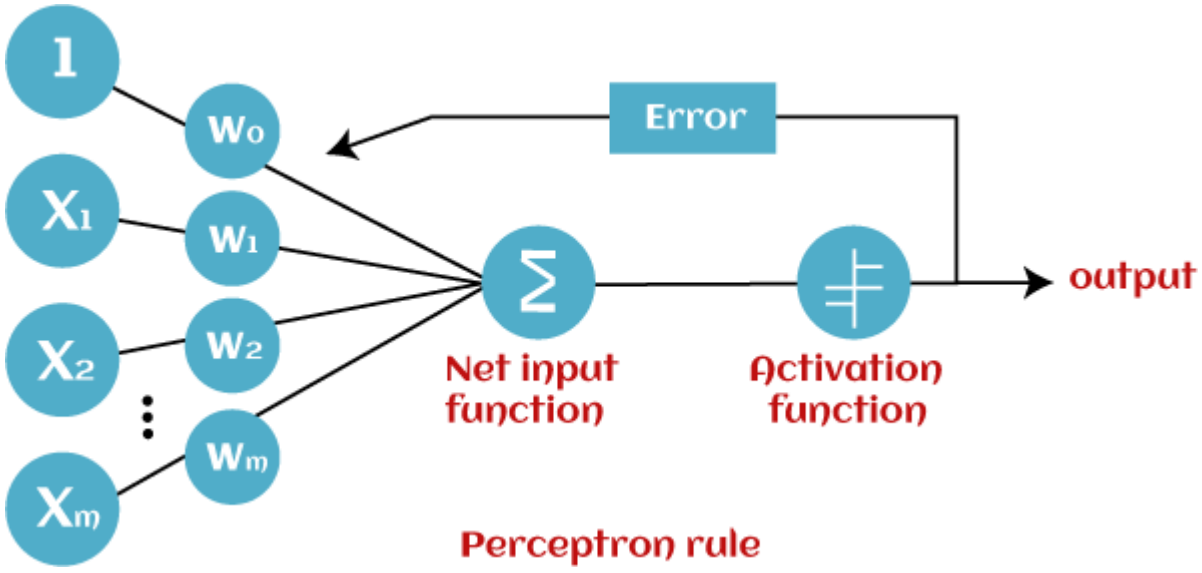
4.3. Experiment No. 3

Aim: Program using Perceptron Neural Network to recognize even and odd numbers. Given numbers are in ASCII from 0 to 9

Objective: To learn about Perceptron Neural Network and its function.

Theory:

Perceptron is a machine learning algorithm which mimics how a neuron in the brain works. It is also called a 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 perceptron laid out in two or more layers. Input to different perceptron in a particular layer will be fed from the previous layer by combining them with different weights.



Types of Perceptron models:

Single Layer Perceptron model: One of the easiest ANN (Artificial Neural Networks) types consists of a feed-forward network and includes a threshold transfer inside the model. The main objective of the single-layer perceptron model is to analyze the linearly separable objects with binary outcomes. A Single-layer perceptron can learn only linearly separable patterns.

Multi-Layered Perceptron model: It is mainly similar to a single-layer perceptron model but has more hidden layers.

Forward Stage: From the input layer in the on stage, activation functions begin and terminate on the output layer.

Backward Stage: In the backward stage, weight and bias values are modified per the model's requirement. The backstage removed the error between the actual output and demands originating backward on the output layer. A multilayer perceptron model has a greater processing power and can process linear and non-linear patterns. Further, it also implements logic gates such as AND, OR, XOR, XNOR, and NOR.

Algorithm:

Inputs:

x: input features, representing an image of a number

y: binary target class label (0 for even, 1 for odd)

w: weight vector

b: bias term

alpha: learning rate

num_iterations: number of iterations to run gradient descent

Outputs:

w: the learned weight vector

b: the learned bias term

Steps:

Initialize the weight vector w and bias term b to zero.

For each iteration of gradient descent:

For each training example (x, y):

Compute the predicted output y_hat =1 if w* x+b>0, else y_hat =0.

Update the weight vector and bias term using the following formulas:

Ww = w + alpha * (y - y_hat) * x

b =b + alpha * (y - y_hat)

Return the learned weight vector w and bias term b.

Applications:

1. A multi-layered perceptron model can be used to solve complex non-linear problems.
2. It works well with both small and large input data.

3. It helps us to obtain quick predictions after the training.
4. It helps to obtain the same accuracy ratio with large as well as small data.

Input:

6

Output:

even

Conclusion:

In this way have successfully implemented Perceptron Neural Network to recognise even

and odd numbers. Given numbers are in ASCII form 0 to 9.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform Perceptron Neural Network to identify Even and Odd number for ASCII number.

Questions:

1. What is Perception Neural Network?
2. What are types of Perception Model?
3. What are the characteristics of perception?
4. What do you understand by perceptron learning algorithm?
5. What do you mean by a neural network?

4.4. Experiment No. 4

Aim:

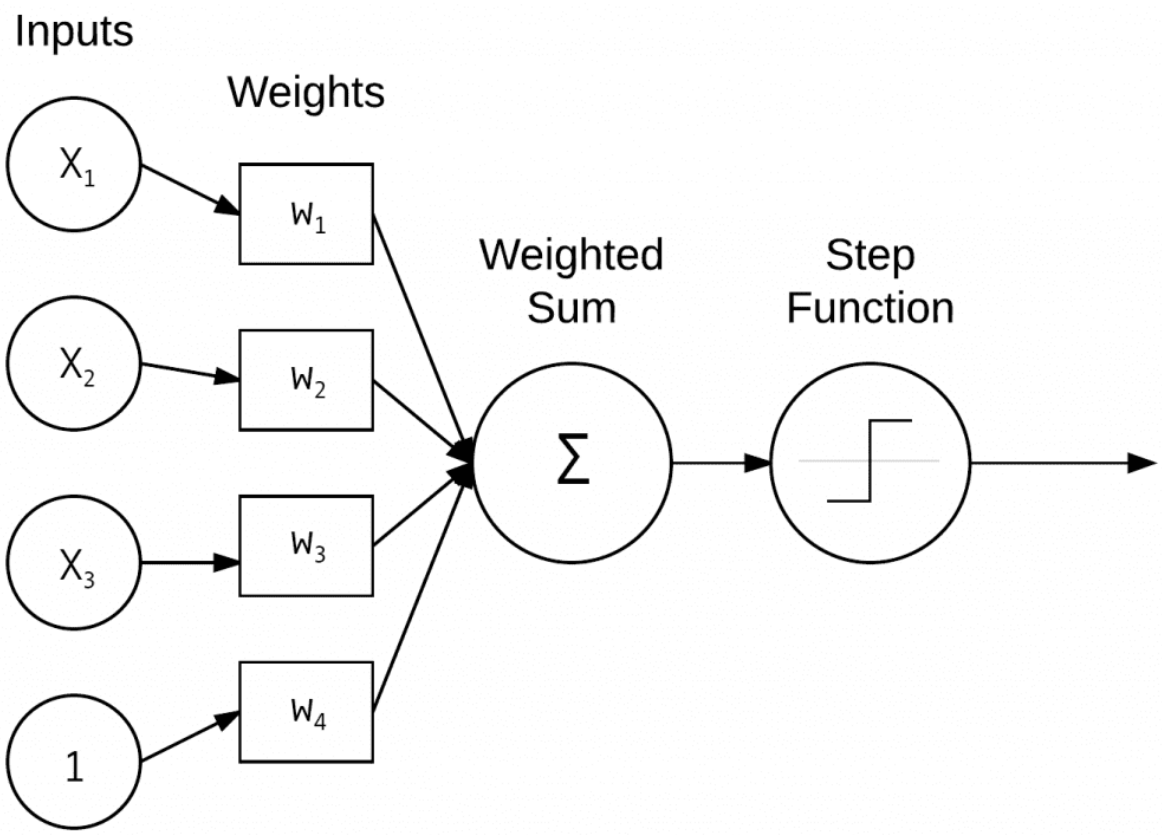
With a suitable example demonstrate the perceptron learning law with its decision regions using python. Give the output in graphical form.

Objective: To learn perceptron laws with decision regions.

Theory: Neural networks are a branch of —Artificial Intelligence". Artificial Neural Network is a system loosely modeled based on the human brain. Neural networks are a powerful technique to solve many real world problems. They have the ability to learn from experience in order to improve their performance and to adapt themselves to changes in the environment. In addition to that they are able to deal with incomplete information or noisy data and can be very effective especially in situations where it is not possible to define the rules or steps that lead to the solution of a problem. In a nutshell a Neural network can be considered as a black box that is able to predict an output pattern when it recognizes a given input pattern. Once trained, the neural network is able to recognize similarities when presented with a new input pattern, resulting in a predicted output pattern.

$$\text{Sum} = f(\sum wi*xi + b)$$

Where b represents bias value, x_i is the input and w_i is the weight of ith neuron (node).



Algorithm:

Perceptron Learning Algorithm: The perceptron learning rule was originally developed by Frank Rosenblatt in the late 1950s. Training patterns are presented to the network's inputs; the output is computed. Then the connection weights are modified by an amount that is proportional to the product of the difference between the actual output, y, and the desired output, d, and the input pattern, x. The algorithm is as follows:

1. Initialize the weights and threshold to small random numbers.
2. Present a vector x to the neuron inputs and calculate the output.
3. Update the weights according to: where d is the desired output, t is the iteration number, and eta is the gain or step size, where 0.0<n< 1.04.

$w_j(t+1) = w_j(t) + \eta(d-y)x$

Repeat steps 2 and 3 until:
1. the iteration error is less than a user-specified error threshold.
2. a predetermined number of iterations have been completed.

Applications:

1. Perceptron is a machine learning algorithm for supervised learning of binary classifiers.
2. In Perceptron, the weight coefficient is automatically learned.
3. Initially, weights are multiplied with input features, and the decision is made whether the neuron is fired or not.
4. The activation function applies a step rule to check whether the weight function is greater than zero.
5. The linear decision boundary is drawn, enabling the distinction between the two linearly separable classes +1 and -1.

Conclusion:

In this way have successfully implemented perceptron learning law with its decision regions using python.

Outcome:

Upon completion of this experiment, students will be able to:
Experiment level outcome (ELO1): Demonstrate the perceptron learning law with its decision regions.

Questions:

1. What is the learning algorithm?
2. What is the Perception learning algorithm?
3. What are the primary components of a perceptron?
4. What are the types of perception model?

4.5 **Experiment No. 5**

Aim:

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

Objective:

To learn the concept of Bidirectional Associative Memory.

Theory:

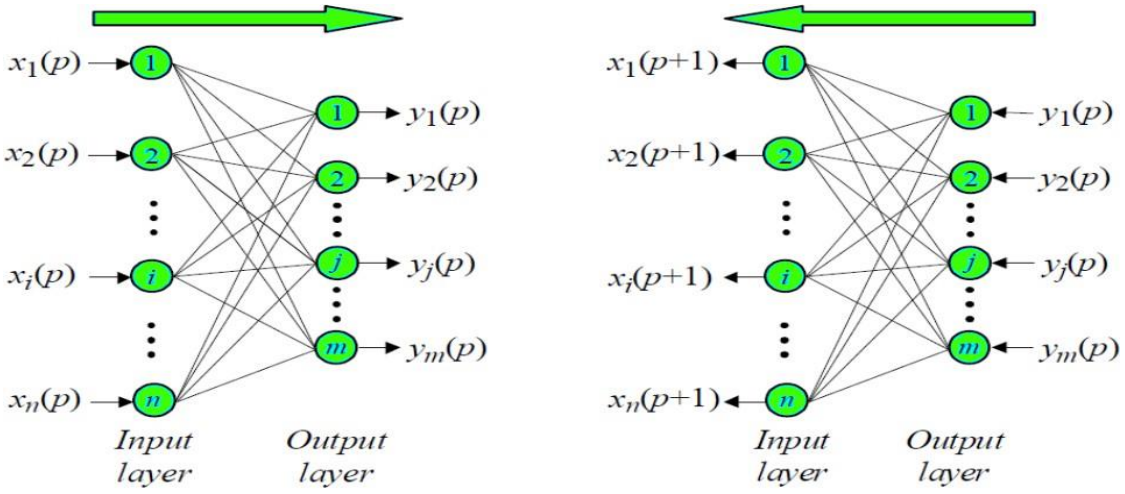
Bidirectional Associative Memory (BAM)

Bidirectional Associative Memory (BAM) is a supervised learning model in Artificial Neural Network. This is ***hetero-associative memory***, for an input pattern, it returns another pattern which is potentially of a different size. This phenomenon is very similar to the human brain. Human memory is necessarily associative. It uses a chain of mental associations to recover a lost memory like associations of faces with names, in exam questions with answers, etc. In such memory associations for one type of object with another, a Recurrent Neural Network (RNN) is needed to receive a pattern of one set of neurons as an input and generate a related, but different, output pattern of another set of neurons.

Why BAM is required?

The main objective to introduce such a network model is to store hetero-associative pattern pairs. This is used to retrieve a pattern given a noisy or incomplete pattern.

BAM Architecture: When BAM accepts an input of ***n***-dimensional vector ***X*** from set ***A*** then the model recalls ***m***-dimensional vector ***Y*** from set ***B***. Similarly when ***Y*** is treated as input, the BAM recalls ***X***.Class Template:



(*a*) Forward direction. (*b*) Backward direction.

Algorithm:

Step 0: Initialize the weights to store p vectors. Also initialize all the activations to zero.

Step 1: Perform Steps 2-6 for each testing input.

Step 2: Ser the activations of X layer to current input pattern, i.e., presenting the input pattern x to X layer similarly presenting the input pattern y to Y layer. Even though it is bidirectional memory, at one time step, signals can be sent from only one layer. So, either of the input patterns may be the zero vector

Step 3: Perform Steps 4-6 when the activations are not converged.

Step 4: Update the activations of units in the Y layer. Calculate the net input,

$$y_{inj} = \sum_{i=1}^{n} x_i w_{ij}$$

Applying activations, we obtain

$$y_j = f(y_{inj})$$

Send this signal to the X layer.

Step 5: Update the activations of units in X layer. Calculate the net input,

$$x_{ini} = \sum_{j=1}^{m} y_j w_{ij}$$

Applying activations, we obtain

$$x_i = f(x_{ini})$$

Send this signal to the Y layer.

Step 6: Test for convergence of the net. The convergence occurs if the activation vectors x and y reach equilibrium. If this occurs then stop, Otherwise, continue.

Input: Students to enter the input pattern

e.g. Weight matrix:

```
[[4 0 4]
 [4 0 4]
 [0 4 0]
 [0 4 0]
 [4 0 4]
 [4 0 4]]
```

Output:

```
Testing for input patterns: Set A
Output of input pattern 1
[[1]
 [1]
 [1]]
Output of input pattern 2
[[-1]
 [-1]
 [-1]]
Output of input pattern 3
[[ 1]
 [-1]
 [ 1]]
Output of input pattern 4
[[-1]
 [ 1]
 [-1]]
Testing for target patterns: Set B
Output of target pattern 1
[[1]
 [1]
 [1]
 [1]
 [1]]

Output of target pattern 2
[[-1]
 [-1]
 [-1]
 [-1]
 [-1]
```

```
 [-1]]


Output of target pattern 3
[[ 1]
 [ 1]
 [-1]
 [-1]
 [ 1]
 [ 1]]


Output of target pattern 4
[[-1]
 [-1]
 [ 1]
 [ 1]
 [-1]
 [-1]]
```

Conclusion:

We have successfully implemented python Program for Bidirectional Associative Memory with two pairs of vectors.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Design various applications for storing and retrieving heterogeneous pattern pairs.

Questions:

1. What is associative learning in neural network?
2. Can data be stored directly in associative memory?
3. What are the different types of associative networks?
4. What is a node in associative networks?
5. The bidirectional associative memory is similar in principle to which model?

4.6 Experiment No. A7

Aim:

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

Objective: To learn about Forward Propagation and Backpropagation in ANN

Theory:

Forward propagation:

Forward propagation is where input data is fed through a network, in a forward direction, to generate an output. The data is accepted by hidden layers and processed, as per the activation function, and moves to the successive layer. The forward flow of data is designed to avoid data moving in a circular motion, which does not generate an output.

Back propagation:

Backward Propagation is the process of moving from right (output layer) to left (input layer). Forward propagation is the way data moves from left (input layer) to right (output layer) in the neural network. A neural network can be understood by a collection of connected input/output nodes. The accuracy of a node is expressed as a loss function or error rate. Backpropagation calculates the slope of a loss function of other weights in the neural network.

Algorithms:

Inputs:

X: a training example input vector
w: the weight matrix for the network
b: the bias vector for the network
Outputs:
a_L: the output of the final layer of the network
Steps:
Set a_0 = X, the input vector for the network.
For each layer | in the network, compute the weighted input z_I for that layer:
Z=w*a-1+b
Apply the activation function g to the weighted input for each layer to compute the activation a_l:
a= g(Z)
Repeat steps 2 and 3 for all layers in the network, up to the final layer L.
Return a_L as the output of the network for input X.

Applications:

Applications of Feedforward Neural Networks and back propagation:

- Pattern recognition.
- Classification tasks.
- Regression analysis.
- Image recognition.
- Computer vision
- Speech recognition
- Natural language processing
- Robotics and even medical diagnosis

Input:

1 0 1

Output:

1

Conclusion:

We have successfully implemented Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Perform predictions for specific application, such as pattern recognition or data classification with better accuracy by implementing Forward Propagation and Back Propagation.

Questions:

1. What is Forward Propagation and Back Propagation?
2. What are the two types of Back Propagation Network?
3. What is the purpose of Forward Propagation?
4. What is Formula for Back Propagation?
5. How is error calculated in Forward Propagation?

4.7. Experiment No. B-1

Aim:

Write a python program to show back propagation network for XOR function with Binary Input and Output.

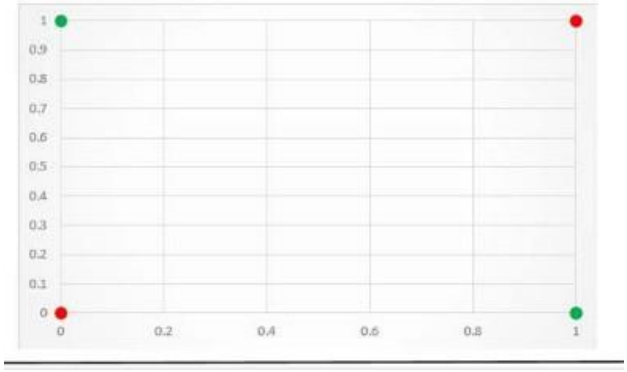
Theory:

Solving XOR problem with Back Propagation Algorithm

XOR or Exclusive OR is a classic problem in Artificial Neural Network Research. An XOR function takes two binary inputs (0 or 1) & returns True if both inputs are different & False if both inputs are same.

Input 1	Input 2	Output
0	0	0
1	1	0
1	0	1
0	1	1

On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s, known as perceptrons. A limitation of this architecture is that it is only capable of separating data points with a single line. This is unfortunate because the XOR inputs are not linearly separable. This is particularly visible if you plot the XOR input values to a graph. As shown in the figure, there is no way to separate the 1 and 0 predictions with a single classification line.



Solution

The backpropagation algorithm begins by comparing the actual value output by the forward propagation process to the expected value and then moves backward through the network, slightly adjusting each of the weights in a direction that reduces the size of the error by a small degree. Both forward and back propagation are re-run thousands of times on each input combination until the network can accurately predict the expected output of the possible inputs using forward propagation.

Algorithm:

Step1: Import the required Python libraries
Step2: Define Activation Function : Sigmoid Function
Step3: Initialize neural network parameters (weights, bias) and define model hyperparameters (number of iterations, learning rate)
Step4: Forward Propagation
Step5: Backward Propagation
Step6: Update weight and bias parameters
Step7: Train the learning model
Step8: Plot Loss value vs Epoch
Step9: Test the model performance

Input:

input vector {x} : (l{x_1, {x_2})

Output:

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

Conclusion:

Hence, the model predicted output for each of the test inputs are exactly matched with the XOR logic gate conventional output, according to the truth table and the cost function is also continuously converging.

Outcome:

Upon completion of this experiment, students will be able to:

Experiment level outcome (ELO1): Students will be able to model a neural network to separate linearly non-seperable input patterns using decision plane.

Questions:

1. What is Back propagataion for XOR?
2. How do you solve XOR with a neural network?
3. How can we design a neural network that acts as an XOR gate?
4. What are the properties of XOR gate?
5. What is a real life example of XOR?

4.8. Experiment No. B-4

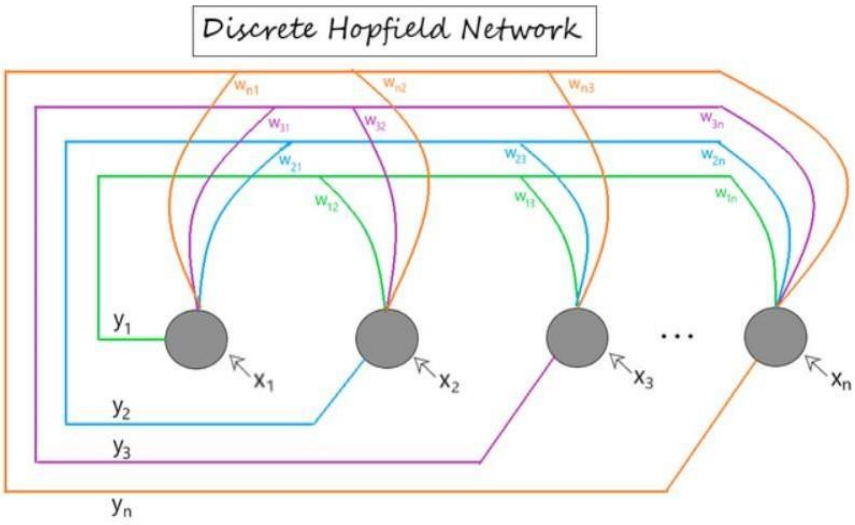
Aim:

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

Theory:

Hopfield neural network was invented by Dr. John J. Hopfield in 1982. It consists of a single layer which contains one or more fully connected recurrent neurons. The Hopfield network is commonly used for auto-association and optimization tasks. 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: A Hopfield network which operates in a discrete line fashion or in other words, it can be said the input and output patterns are discrete vector, which can be either binary 0,1 or bipolar +1,—1 in nature. The network has symmetrical weights with no self-connections i.e., wij = wji and wii = 0.



Discrete Hopfield Network Architecture

Architecture: Following are some important points to keep in mind about discrete Hopfield network:

- This model consists of neurons with one inverting and one non-inverting output.
- The output of each neuron should be the input of other neurons but not the input of self.
- Weight/connection strength is represented by wij.

- Connections can be excitatory as well as inhibitory. It would be excitatory, if the output of the neuron is same as the input, otherwise inhibitory.
- Weights should be symmetrical, i.e. wij = wji

The output from Y1 going to Y2, Yi and Yn have the weights w12, w1i and win respectively. Similarly, other arcs have the weights on them.

Training Algorithm:

During training of discrete Hopfield network, weights will be updated. As we know that we can have the binary input vectors as well as bipolar input vectors. Hence, in both the cases, weight updates can be done with the following relation.

Case 1 – Binary input patterns

For a set of binary patterns s p, p = 1 to P
Here, s p= s1p , s2p ,..., sip,..., snp

Weight Matrix is given by,

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

Case 2 – Bipolar input patterns

For a set of binary patterns s p, p = 1 to P
Here, s p = s1 p, s2 p..., si p..., sn p

Weight Matrix is given by

$$w_{ij} = \sum_{p=1}^{P}[s_i(p)][s_j(p)] \quad for\ i \neq j$$

Testing Algorithm:

Step 1 – Initialize the weights, which are obtained from training algorithm by using Hebbian principle.

Step 2 – Perform steps 3-9, if the activations of the network is not consolidated.

Step 3 – For each input vector X, perform steps 4-8.

Step 4 – Make initial activation of the network equal to the external input vector X as follows:

$$y_i = x_i \quad for\, i = 1\, to\, n$$

Step 5 – For each unit Yi, perform steps 6-9.

Step 6 – Calculate the net input of the network as follows :

$$y_{ini} = x_i + \sum_j y_j w_{ji}$$

Step 7 – Apply the activation as follows over the net input to calculate the output :

Input:

[x1 , x2 , ... , xn] -> Input to the n given neurons.

[y1 , y2 , ... , yn] -> Output obtained from the n given neurons

Wij -> weight associated with the connection between the ith and the jth neuron.

Output:

Finite distinct Output generally of two types: Binary (0/1) Bipolar (-1/1)

Conclusion: Thus We have successfully design a Hopfield Network which stores 4 vectors.

Questions:

1. What is the storage capacity of the Hopfield network?
2. What are the different types of Hopfield neural networks?
3. What is the output of a Hopfield network?
4. What is the structure of the Hopfield network?
5. How many patterns can a Hopfield network of N neurons store?

4.9. Experiment No. C-1

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

Theory:

What is TensorFlow?

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.It was developed by the Google Brain team for internal Google use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released the updated version of TensorFlow, named TensorFlow 2.0, in September 2019. It can be used in a wide variety of programming languages, including Python, JavaScript, C++, and Java. This flexibility lends itself to a range of applications in many different sectors.

What is Pytorch?

PyTorch is an open source machine learning (ML) framework based on the Python programming language and the Torch library. Torch is an open-source ML library used for creating deep neural networks and is written in the Lua scripting language. It's one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment. The framework supports over 200 different mathematical operations. It’s popularity continues to rise, as it simplifies the creation of artificial neural network models. PyTorch is mainly used by data scientists for research and artificial intelligence (Al) applications.

What is regression?

Machine Learning Regression is a technique for investigating the relationship between independent variables or features and a dependent variable or outcome. It’s used as a method for predictive modelling in machine learning, in which an algorithm is used to predict continuous outcomes. 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.

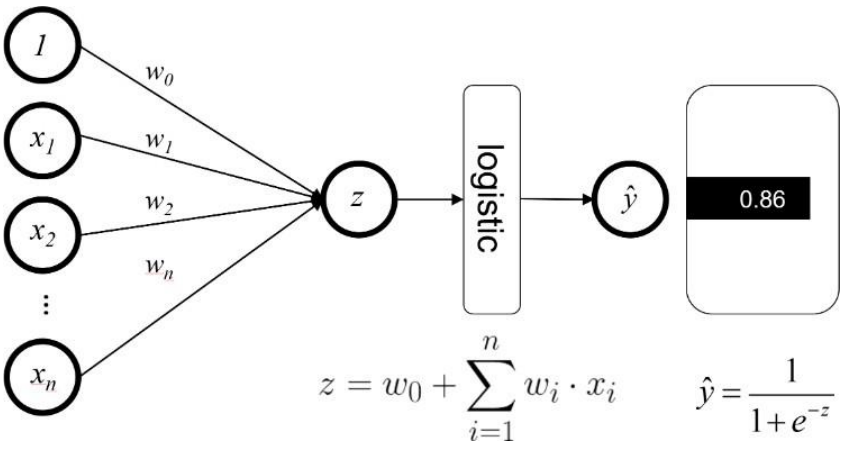
What is neural network?

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.

Logistic Regression:

Logistic regression uses probabilities to distinguish inputs and thereby puts them into separate bags of output classes. To better understand how this process works, let’s look at an example. Consider a case where you want to sketch a relation between your basketball shot’s accuracy and the distance you shoot from. On the whole, it’s about predicting whether you make the basket or not. Let’s

suppose you're going to predict the answer using linear regression. The relation between the win (y) and distance (x) is given by a linear equation, y = mx + c. As a prerequisite, you played for a month, jotted down all the values for x and y, and now you insert the values into the equation. This completes the training phase. Later, you want to estimate the possibility of making the shot from a specific distance. You note the value x and pass it to the trained math equation described above. It will now be a static equation, i.e. y = (trained_m)x + (trained_c).



Algorithm:

Step 1: Importing necessary modules
Step 2: Loading and preparing the mnist data set
Step 3: Setting up hyperparameters and data set parameters
Step 4: Shuffling and batching the data
Step 5: Initializing weights and biases
Step 6: Defining logistic regression and cost function
Step 7: Defining optimizers and accuracy metrics
Step 8: Optimization process and updating weights and biases
Step 9: The training loop
Step 10: Testing model accuracy using the test data

Input:
MNIST dataset e.g. Iris Dataset

Output:
Accuracy (Change of cost over the epochs)

Conclusion: We have successfully implemented a Neural Network with TensorFlow/Pytorch and evaluation of logistic regression using tensorFlow.

Experiment Level Outcome (ELO1): Students will be able to train the neural network and evaluate the performance using Tensorflow.

Questions:

1. What is One-hot Encoder?
2. How do you train a neural network for regression?
3. How to train neural network with TensorFlow?
4. What is neural network regression in ML?
5. Which neural network is best for regression?
6. Is Ann used for classification or regression?

4.10. Experiment No. C-2

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

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 robust test 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 is a widely used dataset of handwritten digits that contains 60,000 handwritten digits for training a machine learning model and 10,000 handwritten digits for testing the model. It was introduced in 1998 and has become a standard benchmark for classification tasks. It is also called the “Hello, World” dataset as it’s very easy to use. MNIST was derived from an even larger dataset, the NIST Special Database 19 which not only contains digits but also uppercase and lowercase handwritten letters. In the MNIST dataset each digit is stored in a grayscale image with a size of 28x28 pixels.

Tensorflow: Tensorflow is an open-source library, and we use TensorFlow to train and develop machine learning models.

Keras: It is also an open-source software library and a high-level TensorFlow API. It also provides a python interface for Artificial Neural Networks.

Pytorch: An open-source ML framework based on Python Programming Language and torch Library.

Implementation:

Dataset: To build this application, we use the MNIST dataset. This dataset contains images of digits from 0 to 9. All these images are in greyscale. There are both training images and testing images. This dataset contains about 60000 training images which are very large and about 10000 testing images. All these images are like small squares of 28 x 28 pixels in size. These are handwritten images of individual digits. Importing Libraries: Import all the required libraries before writing any code. In the beginning, | had already mentioned all the requirements for building the application. So import those libraries. From PIL library import ImageGrab and Image.

Building Model using Tensorflow: To build the model first we need to import some libraries from TensorFlow Keras. We have to import keras from TensorFlow and then import the dataset that we are going to use to build our application now. That is the MNIST dataset. Then import the sequential model, and some layers like Dense, Dropout, Flatten Conv2D, MaxPooling2D, and finally import the backend. After importing all the required libraries, split the dataset into

train and test datasets. Reshape training set of x and testing set of x. The next step is to convert class vectors to binary class matrices.

Training the Model on Tensorflow: Our next step is to train the model. For that define batch size, the number of classes, and the number of epochs that you want to train your model. next add some layers to the sequential model which we imported before. Then compile the model using categorical cross-entropy loss function, Adadelta optimizer, and accuracy metrics. Finally using x_train,y_train, batch size, epochs, and all train the model. Then save it for later.

Predicting Digit: Now we have to write some code for predicting the digit that we have written. For that define a function predict_class where you provide an image as an argument. First, resize it into the required pixels. Convert the image into grayscale which was previously in RGB. Then reshape and normalize it.

Finally, predict the image using predict method. Building Application: Let us see how to build a user-friendly GUI application for it. We use Tkinter for it. Here we create some space for the user to actually draw the digit and then provide two buttons Recognize and clear. Recognize button is to recognize the digit that is written on the given space and the clear button is to clear the writings on it. Finally, run the main loop to run the application. Observation: When you run the application a window will pop up where you can write the digit. And next, when you click on recognize button, it will recognize the digit you have written with the probability percentage showing how exactly the digit matches with the original one. Here | have written digit 1 and it recognized it as 1 with 17% accuracy.

Input:

Output: [5]

Conclusion: We have successfully implemented MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow.

Experiment Level Outcome (ELO 1): Student will be able to understand and implement MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Questions:

1. What is the purpose of handwritten digit recognition?

2. What is the problem with handwritten recognition?

3. What neural network is used for handwritten digit recognition?

4. Which dataset is used for handwritten digit recognition?

5. What is the full form of MNIST?