Dr. D. Y. Patil Pratishthan's

DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT & RESEARCH

Approved by A.I.C.T.E, New Delhi , Maharashtra State Government, Affiliated to Savitribai Phule Pune University
Sector No. 29, PCNTDA , Nigidi Pradhikaran, Akurdi, Pune 411044. Phone: 020–27654470, Fax: 020-27656566
Website :www.dypiemr.ac.in Email : principal.dypiemr@gmail.com

_____

Department of Artificial Intelligence and Data Science

LAB MANUAL Software Laboratory II Artificial Neural Network (TE) Semester II

Prepared by:
Mrs. Arti Singh

Software Laboratory II

Course Code	Course Name	Teaching Scheme (Hrs./ Week)	Credits
317533	Software Laboratory II	04	02

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 practical problems in the real field using Pattern Recognition, Classification and Optimization

Course Outcomes:

On completion of the course, the 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.

Table of Contents

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 hyperparameter tuning to reduce the overfitting or underfitting problems that might occur. Maintain graphs of comparisons.		
4	●	MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow		

Group A

Lab Assignment No.	1
Title	Write a Python program to plot a few activation functions that are being used in neural networks.
Roll No.	
Class	TE
Date of Completion	
Subject	**Elective II Laboratory Artificial Neural Network**
Assessment Marks	
Assessor's Sign	

Title: To study and understand the concept of Python program to plot a few activation functions that are being used in neural networks.

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

Learning Objective:

- To understand and implement a few activation functions.
- To analyze a few activation functions that are being used in neural networks.

Learning Outcome:

- Students will be able to plot a few activation functions that are being used in neural networks.

Software Required:

- Python

Theory:

Importance of Non-Linearity/Activation Functions:

Activation functions are used to introduce non-linearity. What exactly is non-linearity and why is it important to introduce non-linearity? Let's understand it with an example.

Non-linearity in a neural network simply means a function that is able to classify the class of a function that is divided by a decision boundary that is not linear.

It is very hard to find any example in the real world which follows this linearity and hence we need some functions that can approximate a non-linear phenomenon.

If we take all linear functions then the output is nothing but a constant multiplication by the input.

Similarly in a Neural Network, if we do not take non-linearity into account, then the output is just a summation of inputs and weights. No matter how many layers it had, our Neural Network would behave just like a single-layer perceptron, because summing these layers would give you just another linear function.

Activation Functions

(i) Step Activation Function: The Step activation function is used in the perceptron network. This is usually used in single-layer networks to convert to an output that is binary (0 or 1) or Bipolar (-1 or 1). These are called Binary Step Function and Bipolar Step Function Respectively. Here if the input value to a function is greater than a threshold value then the output is 1 (neuron will fire) otherwise 0 or -1 in the case of Bipolar Step Function (neuron will not fire).

(ii) Sigmoid Functions – The step activation function which is a logic used in perceptron is that if the value is greater than the threshold value then the output is 1 (neuron will fire) otherwise 0 (neuron will not fire) and this logic is used by perceptron is very **harsh.**

For eg: Suppose you want to watch a movie and decide whether you like it or not. Consider that we have only 1 input (x1-critics rating which lies between 0 and 1). Now if the threshold is 0.5 and the critics rating = 0.51 then we would like the movie and if the critic rating = 0.49 then we would dislike the movie. This is what harsh means here.

WHY SIGMOID?

Sigmoid functions are broadly used in Back Propagation Networks because of the relationship between the value of the function and the value of the derivative at that point. This reduces overall computation overload.

There are two types of sigmoidal functions:

1. Binary Sigmoid
2. Bipolar Sigmoid

Binary Sigmoid Function:

This is also known as the logistic sigmoid function. Its range lies between 0 and 1. The Sigmoid function gives the output in probability and it is smoother than the perceptron function. If w(t)x tends to infinity then the output gets close to If w(t)x tends to negative infinity the output gets close to 0.

Binary Sigmoid Function:

This is also known as the logistic sigmoid function. Its range lies between 0 and 1. The Sigmoid function gives the output in probability and it is smoother than the perceptron function. If w(t)x tends to infinity then the output gets close to If w(t)x tends to negative infinity the output gets close to 0.

Graph:
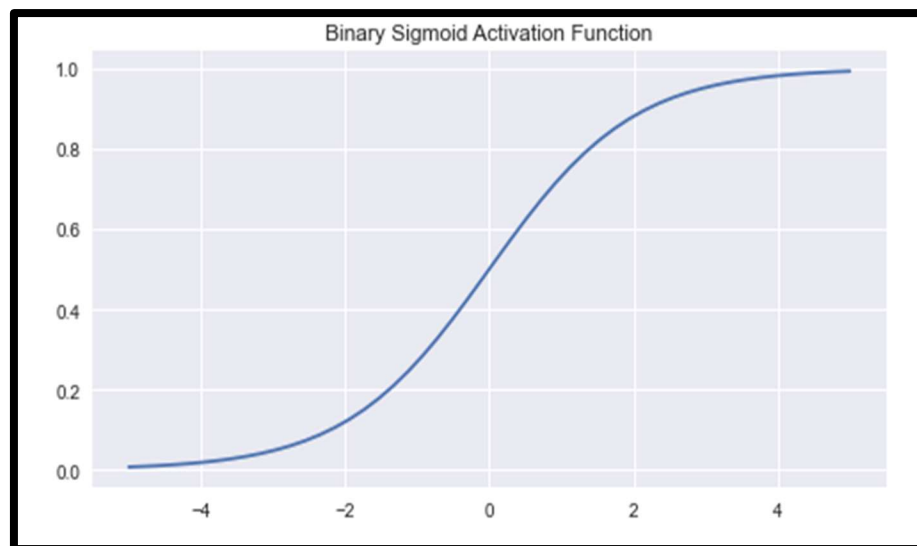
Bipolar Sigmoid Function: This is the function from where the Hyperbolic Tan Function was derived from. Here (lambda) represents the steepness factor. The range of this function is between -1 and 1. For the hyperbolic tangent function, the value of the steepness factor is 2.

Graph:
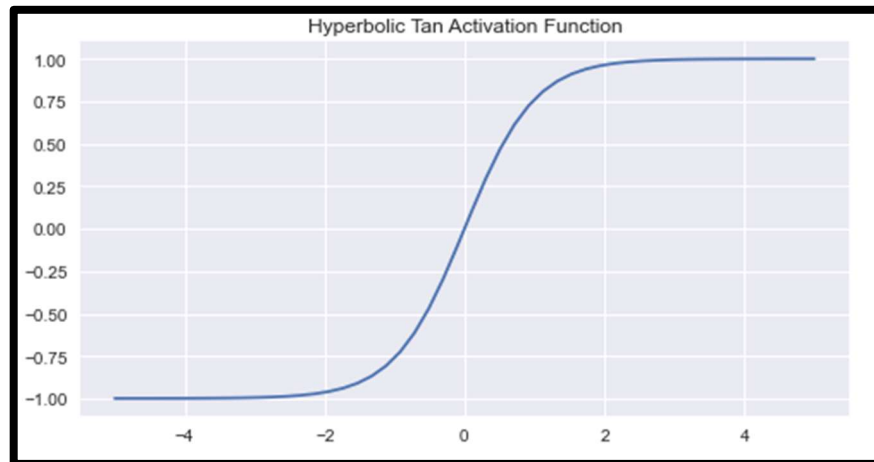
The following image shows the relationship between the Binary and Bipolar Functions and their derivatives:

Activation Functions

- Binary Sigmoid
  $f(x) = 1 / [1 + e^{-x}]$
  $f'(x) = f(x)[1 - f(x)]$
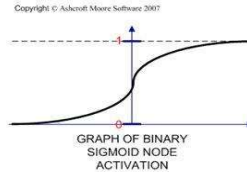
- Bipolar Sigmoid
  $f(x) = -1 + 2 / [1 + e^{-x}]$
  $f'(x) = 0.5 * [1 + f(x)] * [1 - f(x)]$
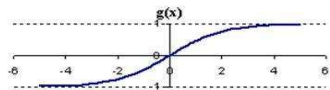
There were various problems with sigmoid and hyperbolic tan activation functions and that's why we need to see a few more activation functions such as ReLu, Leaky ReLu, Elu, etc. We are going to describe them below:

(ii) **RELU (Rectified Linear Unit)**: Some problems with sigmoid and Hyperbolic Tan (tanh) activation functions such as Vanishing Gradient Problem and Computational Expensive Problem.

The RELU activation function returns 0 if the input value to the function is less than 0 but for any positive input, the output is the same as the input. It is also continuous but non-differentiable at 0 and at values less than 0 because its derivative is 0 for any negative input.
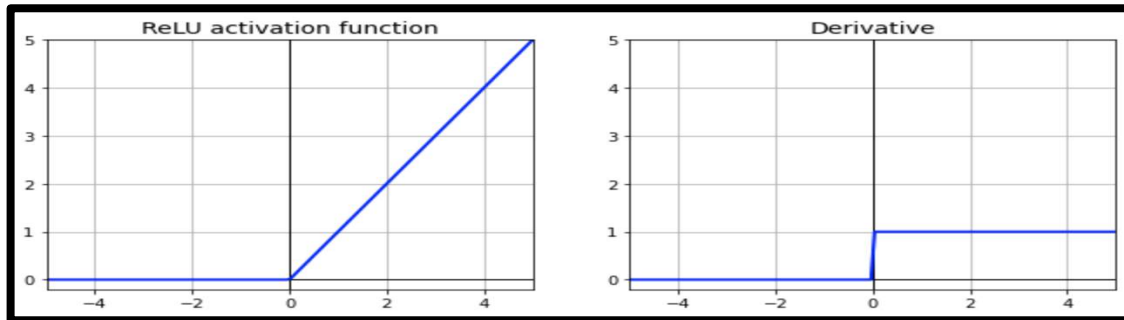
(iii) **Leaky RELU**: in order to solve the DyingRELU problem, people proposed to set the first half of RELU **0.01x** instead of 0.

With Leaky RELU there is a small negative slope so instead of that firing at all, for large gradients, our neurons do output some value and that makes our layer much more optimized too.

(
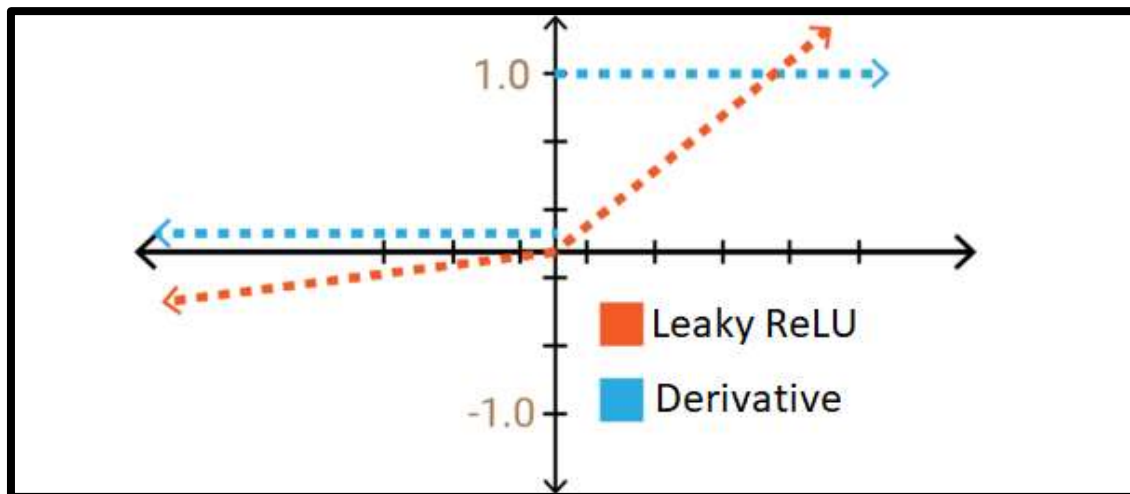
(iv) **ELU (Exponential Linear Unit) function**: It is the same as RELU for positive input, but for negative inputs, ELU smoothes slowly whereas RELU smoothes sharply.

Some of the features of ELU activation are:

● No Dead RELU issues

- The mean of output is close to 0
- One small problem is that it is slightly more computationally intensive
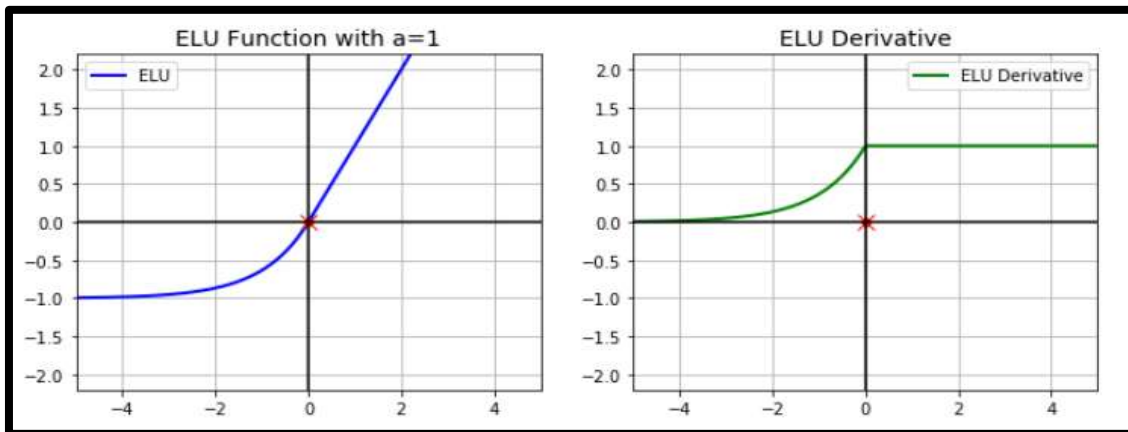
(v) **Softmax Function**: it not only maps our output to [0,1] range but also maps each output in such a way that the total sum is 1. The output of SoftMax is therefore a probability distribution.

It is often used in the final layer of a Neural Network for a multiclass classification problem.

For example, let us suppose that the output of the last layer was {40,30,20} of the following Neural Network.

So when we apply the SoftMax function in the output layer:

$$1st\ neuron\ -\ \frac{e^{40}}{e^{40} + e^{30} + e^{20}}$$

$$2nd\ neuron\ =\ \frac{e^{30}}{e^{40} + e^{30} + e^{20}}$$

$$3rd\ neuron\ =\ \frac{e^{20}}{e^{40} + e^{30} + e^{20}}$$

In conclusion, softmax is used for multiclass classification in the logistic regression model whereas sigmoid is used for binary classification in the logistic regression model.

Graph:
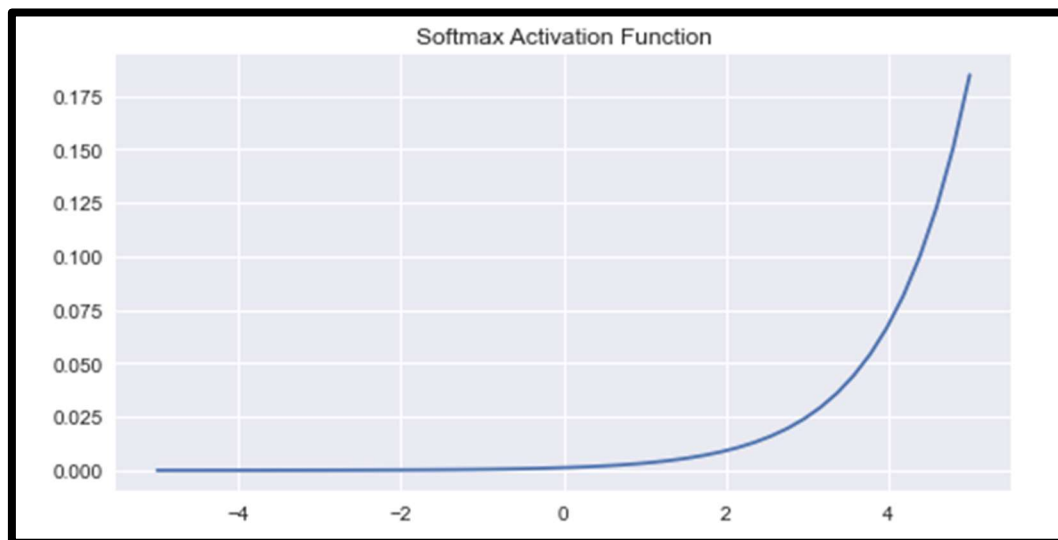
Conclusion: Thus, we have studied……………………………………….
using python.

Lab Assignment No.	2
Title	Generate ANDNOT function using McCulloch-Pitts neural net by a python program.
Roll No.	
Class	TE
Date of Completion	
Subject	**Elective II Laboratory Artificial Neural network**
Assessment Marks	
Assessor's Sign	

Assignment No-02

Title: To study and understand the concept AND NOT function using McCulloch-Pitts neural net by using python.

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

Learning Objective:

- To understand and implement ANDNOT function using McCulloch-Pitts neural net.
- .To analyze McCulloch-Pitts neural net

Learning Outcomes:

- Ability to apply ANDNOT function McCulloch-Pitts neural net.

Software Required:

- Python

Theory:

McCulloch-Pitts Neuron

It is very well known that the most fundamental unit of deep neural networks is called an *artificial neuron/perceptron*. But the very first step towards the *perceptron* we use today was taken in 1943 by McCulloch and Pitts, by mimicking the functionality of a biological neuron.

McCulloch-Pitts Neuron

The first computational model of a neuron was proposed by Warren McCulloch (neuroscientist) and Walter Pitts (logician) in 1943.
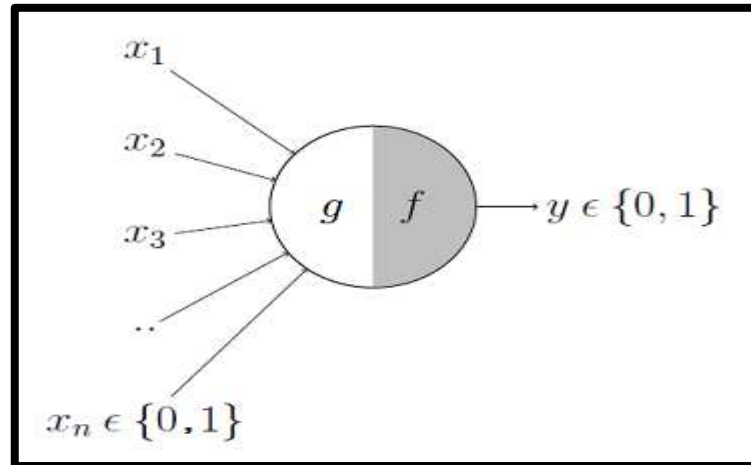
It may be divided into 2 parts. The first part, *g* takes an input (dendrite), performs an aggregation and based on the aggregated value the second part, *f* makes a decision.

Let's suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., {0,1} and my output variable is also boolean {0: Will watch it, 1: Won't watch it}.

- So, *x_1* could be *isPremierLeagueOn* (I like Premier League more)

- *x_2* could be *isItAFriendlyGame* (I tend to care less about the friendlies)

- *x_3* could be *isNotHome* (Can't watch it when I'm running errands. Can I?)

- *x_4* could be *isManUnitedPlaying* (I am a big Man United fan. GGMU!) and so on.

These inputs can either be *excitatory* or *inhibitory*. Inhibitory inputs are those that have maximum effect on the decision-making irrespective of other inputs i.e., if *x_3* is 1 (not home) then my output will always be 0 i.e., the neuron will never fire, so *x_3* is an inhibitory input. Excitatory inputs are NOT the ones that will make the neuron fire on their own but they might fire it when combined together. Formally, this is what is going on:

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

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

We can see that $g(\mathbf{x})$ is just doing a sum of the inputs — a simple aggregation. And *theta* here is called the thresholding parameter. For example, if I always watch the game when the sum turns out to be 2 or more, the *theta* is 2 here. This is called the Thresholding Logic.

Boolean Functions Using M-P Neuron

A lot of boolean decision problems can be cast into this, based on appropriate input variables— like whether to continue reading this post, whether to watch Friends after reading this post etc. can be represented by the M-P neuron.

M-P Neuron: A Concise Representation
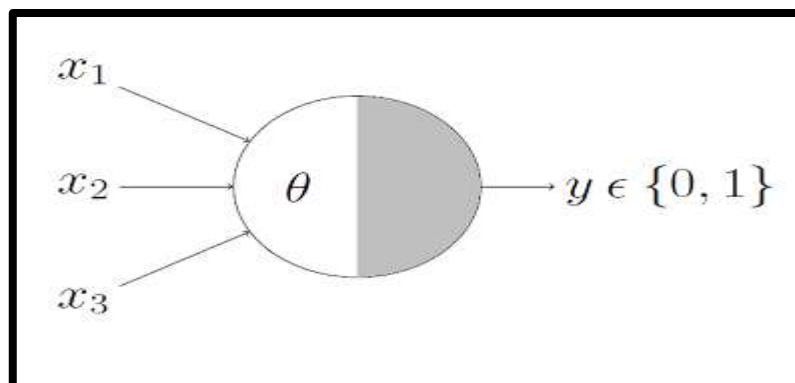
This representation just denotes that, for the boolean inputs x_1, x_2 and x_3 if the $g(x)$ i.e., **sum $\geq$ theta**, the neuron will fire otherwise, it won't.

Question : Implement ANDNOT function using McCulloch-Pitts neuron (cake binary data).

SOLUTION :

In ANDNOT function, response is true if first input is true and second false. For all other variations, response is false.

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

ANDNOT Function:
Truth Table:

X1	X2	Y
1	1	0
1	0	1
0	1	0
0	0	0
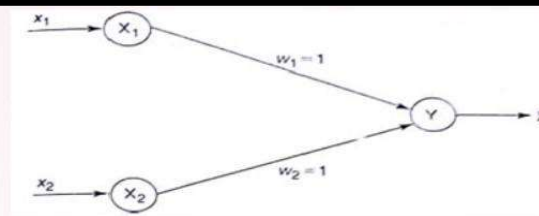
Case 1: Assume that both weights $W1$ and $W2$ are excitatory, i.e.,
W1=W2=1
Then for the four inputs calculate the net input using
$Yin = x1w1 + x2w2$
For inputs
(1, 1), $Yin = 1 \times 1 + 1 \times 1 = 2$
(1, 0), $Yin = 1 \times 1 + 0 \times 1 = 1$
(0, 1), $Yin = 0 \times 1 + 1 \times 1 = 1$
(0, 0), $Yin = 0 \times 1 + 0 \times 1 = 0$

From the calculated net inputs, it is not possible to fire the neuron for input (1, 0) only. Hence, these weights are not suitable :

CASE 2 :
Assume one weight as excitatory and the other as inhibitory
w1 =1, w2=-1

Now calculate the net input. For the inputs
(1, 1), *Yin*= 1 X 1 + 1X -1 = 0
(1, 0), *Yin*= 1 X 1 + 0 X -1 = 1
(0, 1), *Yin* = 0 X 1 + 1 X -1 = -1
(0, 0), *Yin* = 0 X 1 + 0 X -1 = 0

From the calculated net inputs, now it is possible to fire the neuron for input (1, 0) only by fixing a threshold of 1. Thus output can be given by :

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 1 \\ 0 & \text{if } y_{in} < 1 \end{cases}$$

Lab Assignment No.	3
Title	Write a Python Program using Perceptron Neural Network to recognise even and odd numbers. Given numbers are in ASCII form 0 to 9
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 03

Title: Implement digit recognition (even and odd numbers) using perceptron neural network

Problem Statement: Write a Python Program using Perceptron Neural Network to recognize even and odd numbers. Given numbers are in ASCII from 0 to 9

Learning Objective:

- To understand and implement digit recognition.
- To analyze perceptron neural network

Learning Outcomes:

- Ability to recognize even and odd numbers using Perceptron Neural Network problem-solving

Software Required:

- Python

Theory:

ANN has been there in the field of machine learning and Artificial Intelligence for a long time, but recent improvements in computational power and big data is helping them to show how powerful they are. They can be used to solve both supervised and unsupervised, classification and regression problems, computer vision etc. In this article, we shall be implementing an ANN from scratch and applying it to solve a simple problem of detecting digits from 0−9.

Neural Network is similar to logistic regression (perceptron) but with more layers and hidden units. Here's what a single-layer perceptron (logistic regression) looks like.

Here's is what a single-layer perceptron (logistic regression) looks like.
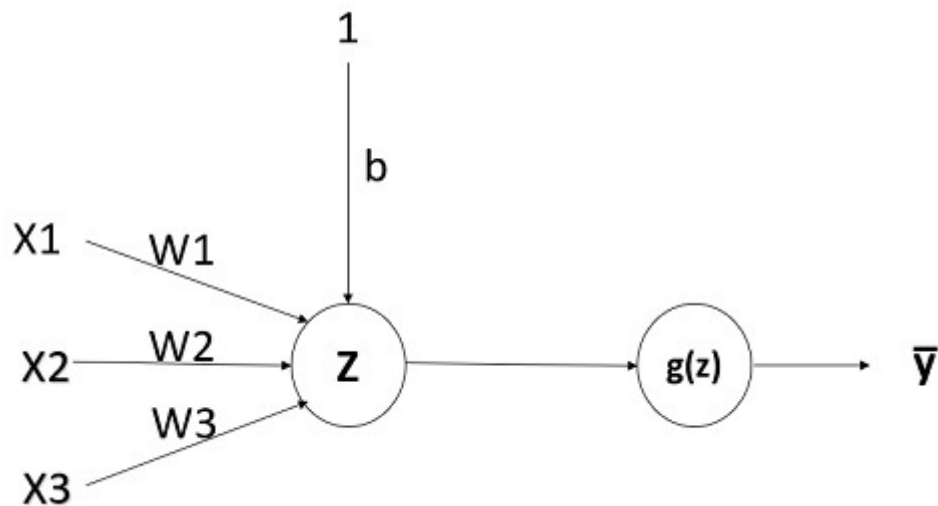
Here x1, x2, x3 are the features of the input, w1, w2, and w3 are the parameters, and b is the bias. Now Z= w1 X1+w2 X2+w3 X3+b. g(z) is the activation function.

There are many available activation functions like sigmoid, tanh, relu, leaky relu etc. If the problem is a classification problem, the most common choice of activation function is sigmoid. The formulae and plots of these activation functions are

Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
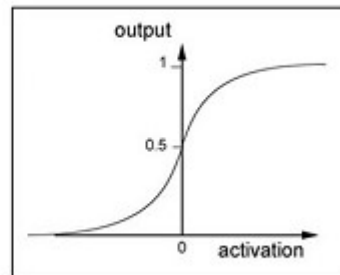
Tanh:

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

Relu (Rectified Linear Unit):

$$\sigma(z) = \max(0, z)$$
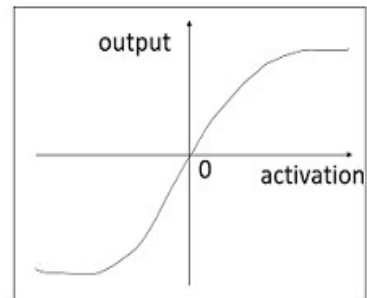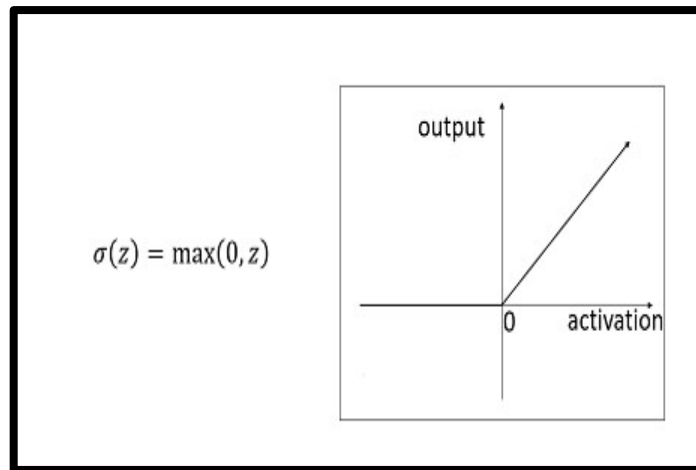
Similar to the perceptron model, a simple one-layer neural network looks

like:
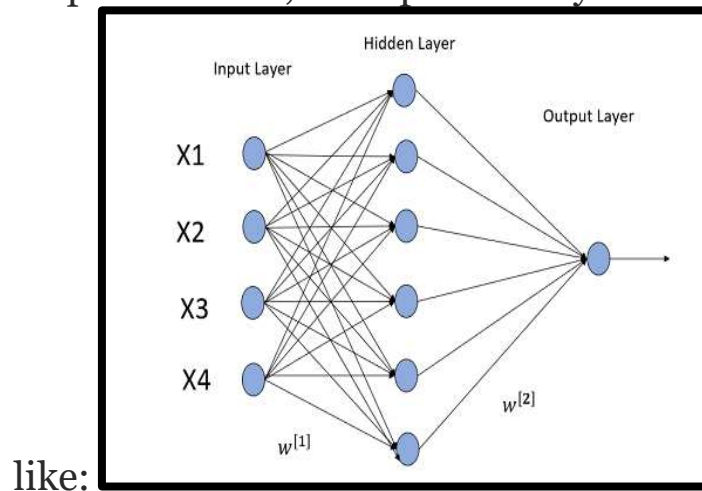
The input layer is where the features of the input features (X) is fed, the Hidden layer will be the weighted sum of the inputs with parameters (W) followed by an activation function. There will always be bias (b) for each

Department of Artificial Intelligence and Data Science, DYPIEMR, Pune

hidden layer. The output of the neural network will be the weighted sum of outputs of the previously hidden layers followed by an activation function. For a two-class classifier problem, the output layer will be the probability that the given training example belongs to the class (c_i).

Similarly, a 4 layered deep neural network looks like this:
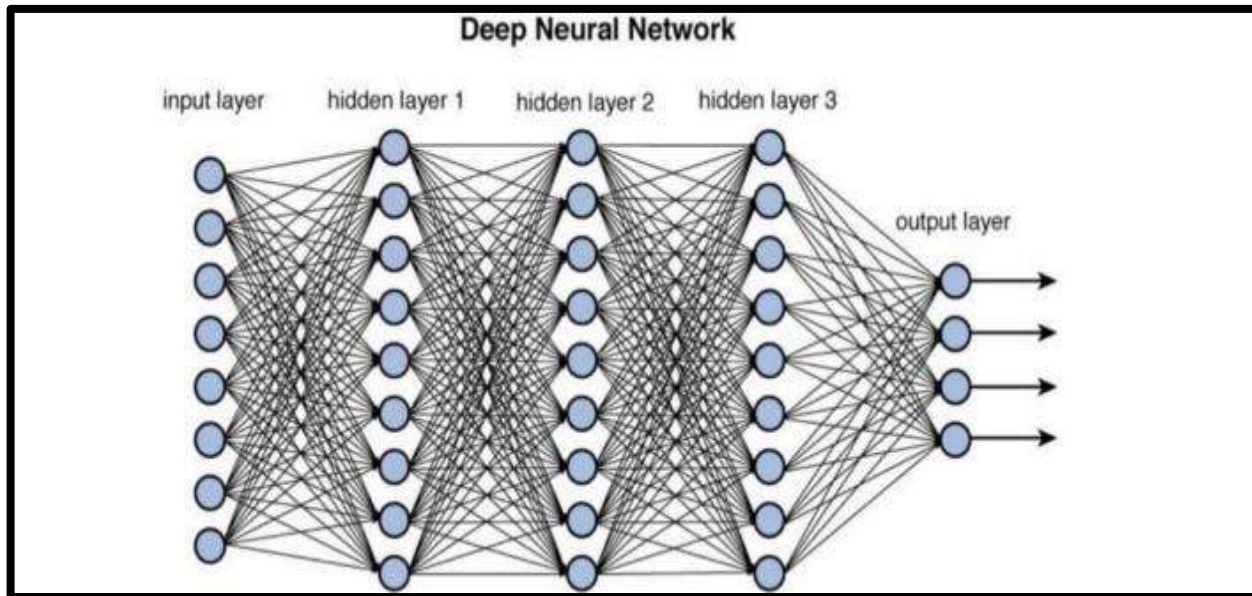
Implementation from Scratch:

Data preprocessing Steps:

1. Importing libraries

2. Load the digits data sets from sklearn. datasets

    3. Let's see the first image in the dataset:

<Figure size 432x288 with 0 Axes>

https://medium.com/machine-learning-algorithms-from-scratch/digit-recognition-from-0-9-using-deep-neural-network-from-scratch-8e6bcf1dbd3

.
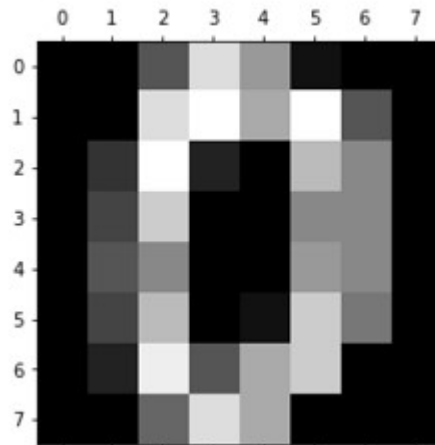
Lab Assignment No.	4
Title	Write a python Program for Bidirectional Associative Memory with two pairs of vectors.
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 04

Title: To study and understand the concept of Bidirectional Associative Memory with two pairs of vectors.

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

Learning Objective:

- To understand the concept of Bidirectional Associative Memory.

- To implement Bidirectional Associative Memory with two pairs of vectors.

Learning Outcome: Students will be able to use Bidirectional Associative Memory with two pairs of vectors.

Software Required: Python

Theory:

Bidirectional associative memory (BAM) is a type of recurrent neural network. BAM was introduced by Bart Kosko in 1988. There are two types of associative memory, auto-associative and hetero-associative. BAM is hetero-associative, meaning given a pattern it can return another pattern which is potential of a different size. It is similar to the Hopfield network in that they are both forms of associative memory. However, Hopfield nets return patterns of the same size.

It is said to be bi-directional as it can respond to inputs from either the input or the output layer.

Topology

A BAM contains two layers of <u>neurons</u>, which we shall denote X and Y. Layers X and Y are fully connected to each other. Once the weights have been established, input into layer X presents the pattern in layer Y, and vice versa.

The layers can be connected in both directions (bidirectional) with the result of the weight matrix sent from the X layer to the Y layer is and the weight matrix for signals sent from the Y layer to the X layer is. Thus, the weight matrix is calculated in both directions.

Procedure

Learning

Imagine we wish to store two associations, A1:B1 and A2:B2.

- A1 = (1, 0, 1, 0, 1, 0), B1 = (1, 1, 0, 0)
- A2 = (1, 1, 1, 0, 0, 0), B2 = (1, 0, 1, 0)

These are then transformed into the bipolar forms:

- X1 = (1, -1, 1, -1, 1, -1), Y1 = (1, 1, -1, -1)
- X2 = (1, 1, 1, -1, -1, -1), Y2 = (1, -1, 1, -1)

From there, we calculate where denotes the transpose. So,

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*.

(a) Forward direction.   (b) Backward direction.
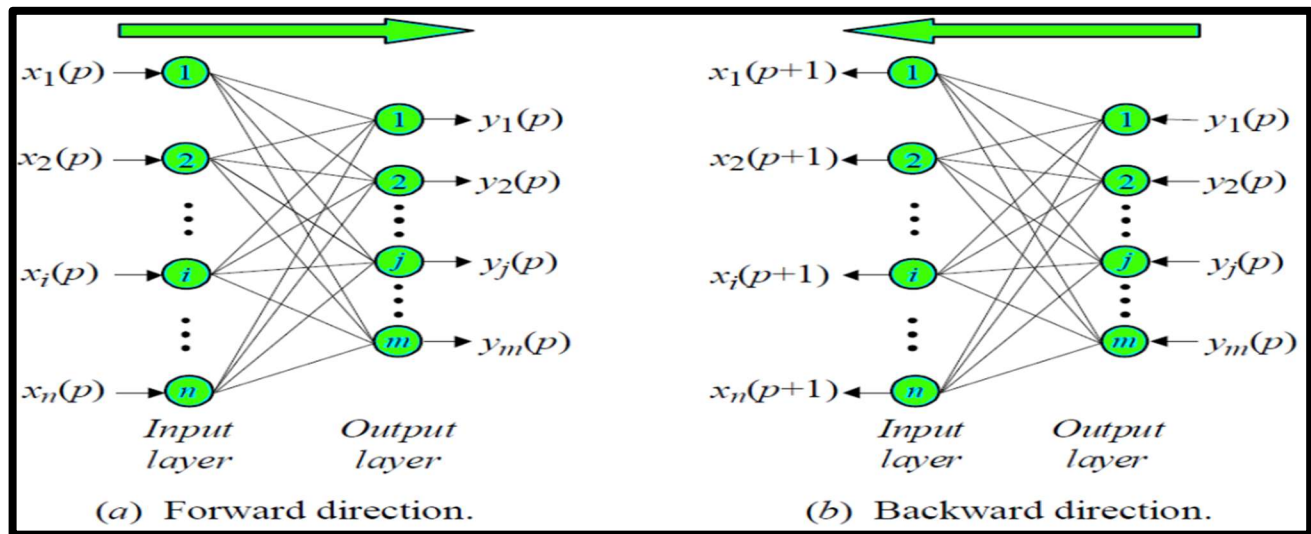
Conclusion: Thus, we have studied use

Lab Assignment No.	5
Title	Implement Artificial Neural Network training process in Python by using Forward Propagation, Back Propagation..
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 5

Title: To illustrate Artificial Neural Network training process in Python

Problem Statement: Implement an Artificial Neural Network training process in Python by using Forward Propagation, and Back Propagation.

Objective:

-
- To understand and implement Artificial Neural Networks
- Analyze time

Outcomes:

- Ability to choose an appropriate problem-solving method and knowledge representation technique.

Software Required:

- Python

Theory: Backpropagation neural network is used to improve the accuracy of <u>neural network</u> and make them capable of self-learning. Backpropagation means "backward propagation of errors". Here error is spread into the reverse direction in order to achieve better performance. Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.

BPN was discovered by Rumelhart, Williams & Honton in 1986. The core concept of BPN is to backpropagate or spread the error from units of the output layer to internal hidden layers in order to tune the weights to ensure lower error rates. It is considered a practice of fine-tuning the weights of neural networks in each iteration. Proper tuning of the weights will make a sure minimum loss and this will make a more robust, and generalizable trained neural network.

$$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$$
(compute gradient)

(error term of the output layer)
$$\delta^{(3)} = a^{(3)} - y$$

Input x   output $\hat{y}$ ← target *y*

$$\delta^{(2)} = \left(W^{(2)}\right)^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$$
(error term of the hidden layer)
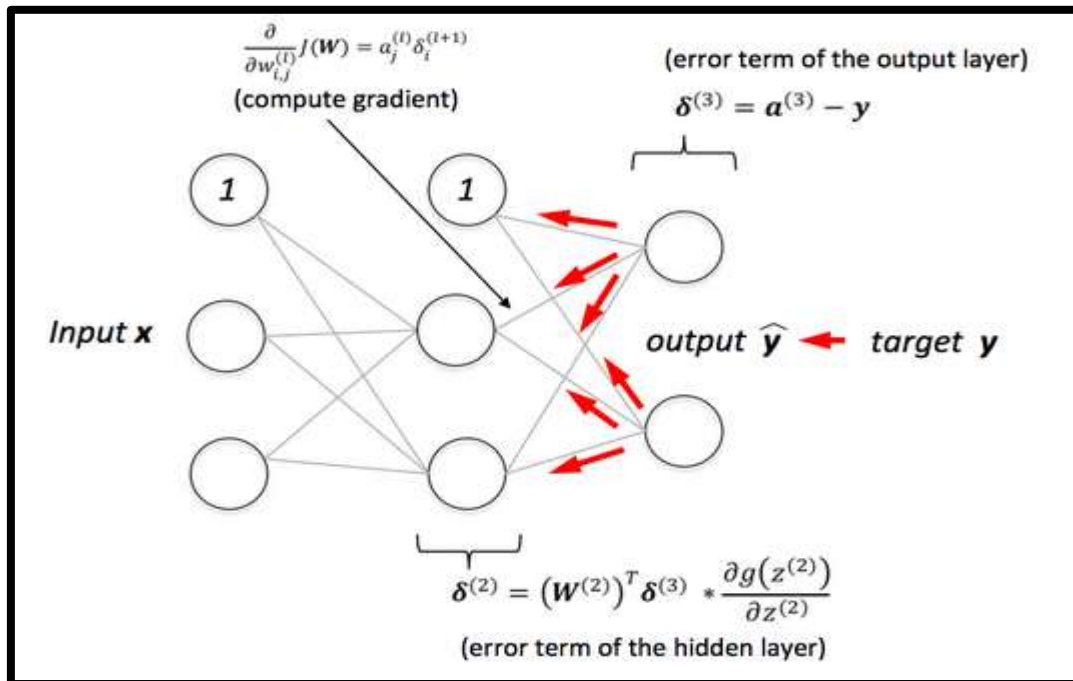
BPN learns in an iterative manner. In each iteration, it compares training examples with the actual target label. target label can be a class label or continuous value. The backpropagation algorithm works in the following steps:

- **Initialize Network:** BPN randomly initializes the weights.
- **Forward Propagate:** After initialization, we will propagate into the forward direction. In this phase, we will compute the output and calculate the error from the target output.
- **Back Propagate Error**: For each observation, weights are modified in order to reduce the error in a technique called the delta rule or gradient descent. It modifies weights in a "backward" direction to all the hidden layers.

  - Forward propagation refers to the storage and calculation of input data which is fea d in forward direction through the network to generate an output. Hidden layers in neural network accepts the data from the input layer, process it on the basis of activation function and pass it to the output layer or the successive layers. Data flows in forward direction so as toa avoid circular shape flow of data which will not generate an output. The network configuration that helps in forward propagation is a known as feed-forward network.

  - You can call it as thetas' first initialization, the period when the network get to know the data for the first time, later we'll use backpropagation to develop thetas in each layer.
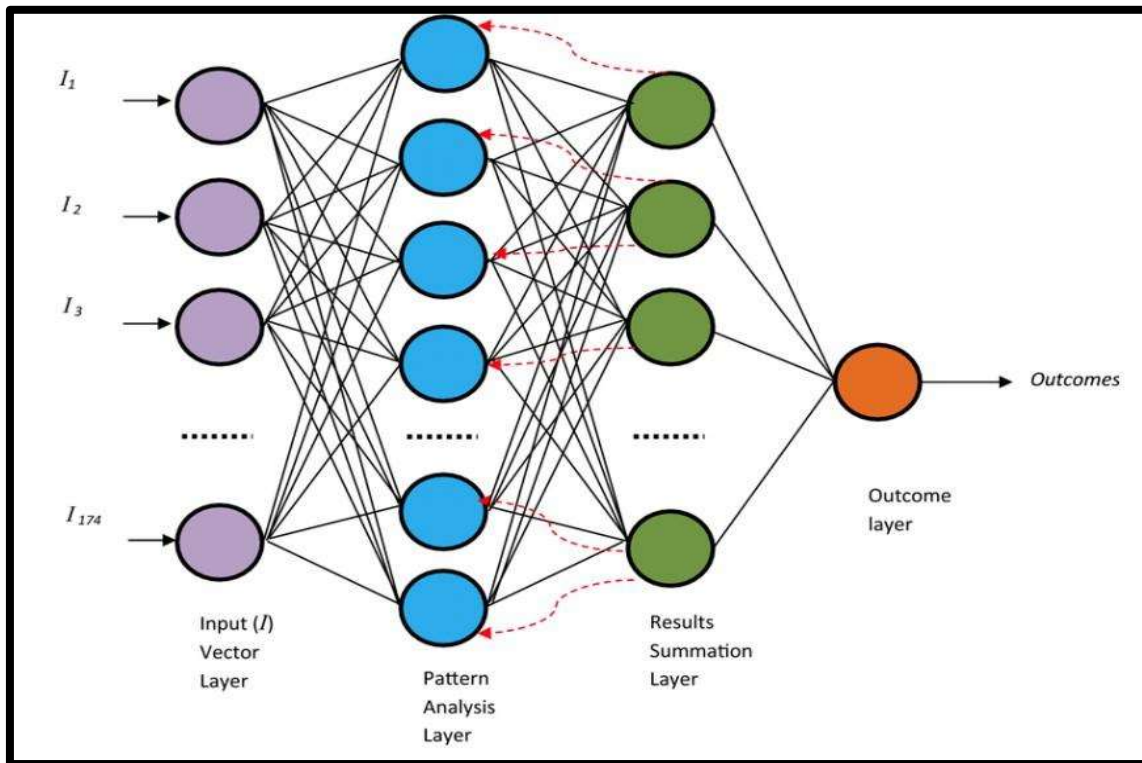
Important Terms

1. The first layer is called the **input layer**. It has all the features you want to feed into the network.
2. From the second layer till the layer before the last layer we call them the **hidden layers**.
3. Finally the last layer is called the **Outcome/ Output layer**.

- The outcome layer has one neuron just for **binary classification**, but you can have more than one in case you have a problem of **multiclass classification**, In this case, you will use the **softmax** function, we might know it later.

linkcode

Steps of Forward Propagation

1. Select carefully the features you want to feed into the network
   - To avoid overfitting / complexity of the model.
   - Reduce the time used to train the model
2. Add the bias term.
3. Each feature will have the same number as thetas as the number of the first layer's neurons, plus the bias.
4. Preactivation: it is a weighted sum of inputs i.e. the linear transformation of weights w.r.t to inputs available. Based on this aggregated sum and activation function the neuron makes a decision whether to pass this information further or not.

5. Activation: the calculated weighted sum of inputs is passed to the activation function. An activation function is a mathematical function which adds non-linearity to the network. There are four commonly used and popular activation functions — **sigmoid, hyperbolic tangent(tanh), ReLU and Softmax.**

Conclusion: Thus, we have implemented

Lab Assignment No.	6
Title	With a suitable example demonstrate the perceptron learning law with its decision regions using python. Give the output in graphical form
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 06

Title: To illustrate perceptron learning law with its decision regions

Problem Statement: Implement perceptron learning law with its decision regions using python. Give the output in graphical form

Objective:

- To understand and implement perceptron learning law
- Analyze decision regions

Outcomes:

- Ability to choose an appropriate problem-solving method and knowledge representation technique.

Software Required:

- Python

Theory: A perceptron, a neuron's computational prototype, is categorized as the simplest form of a neural network. Frank Rosenblatt invented the perceptron at the Cornell Aeronautical Laboratory in 1957. A perceptron has one or more than one inputs, a process, and only one output.

The concept of perceptron has a critical role in machine learning. It is used as an algorithm or a linear classifier to facilitate supervised learning of binary classifiers. Supervised learning is amongst the most researched of learning problems. A supervised learning sample always consists of an input and a correct/explicit output. The objective of this learning problem is to use data with correct labels for

making predictions on future data, for training a model. Some of the common problems of supervised learning include classification to predict class labels.

A linear classifier that the perceptron is categorized as is a classification algorithm, which relies on a linear predictor function to make predictions. Its predictions are based on a combination that includes weights and feature vector. The linear classifier suggests two categories for the classification of training data. This means, if classification is done for two categories, then the entire training data will fall under these two categories.

The perceptron algorithm, in its most basic form, finds its use in the binary classification of data. Perceptron takes its name from the basic unit of a neuron, which also goes by the same name.

• The primary components of a perceptron

1. **Input:** Features are taken as inputs in the perceptron algorithm. Inputs are denoted as x1, x2, x3, x4, .xn – 'x' in these inputs indicates the feature value and 'n' the total occurrences of these features. There is also a special input type, which is called bias.

2. **Weights**: These are values that are calculated during the training of the model. The weights are given an initial value at the start. With every

occurrence of a training error, the values of weights are updated. Weights are represented as w1, w2, w3, w4, ..wn.

3. **Bias:**bias is a special input type. It allows the classifier to move the decision boundary around from its original position to the right, left, up, or down. In terms of algebra, the bias allows the classifier to turn its decision boundary around. The objective of the bias is to shift each point in a particular direction for a specified distance. Bias allows for higher quality and faster model training.

4. **Activation/step function:** Activation or step functions are used to create non-linear neural networks. These functions can change the value of neural networks to 0 or 1. The conversion of value is done to make a data set easy to classify. We can use the step function depending on the value required. Sigmoid function and sign functions can be used for values between 0 and 1 and 1 and -1, respectively.

5. **Weighted summation:** The multiplication of every feature or input value (xn) associated with corresponding weight values (wn) gives us a sum of values that are called weighted summation. Weighted summation is represented as ∑wixi for all i -> [1 to n].

6. **Neurons:** A neural network is composed of a collection of odes or units known as neurons.

7. **Synapse:** The getting neuron can obtain the sign, process the same, and sign the subsequent one. A neuron can send information or signals through the synapse to another adjacent neuron. It processes it and signals the subsequent one. This process in the perceptron algorithm continues until an output signal is generated.

Perceptron Learning Rule

Perceptron Learning Rule states that the algorithm would automatically learn the optimal weight coefficients. The input features are then multiplied with these weights to determine if a neuron fires or not.
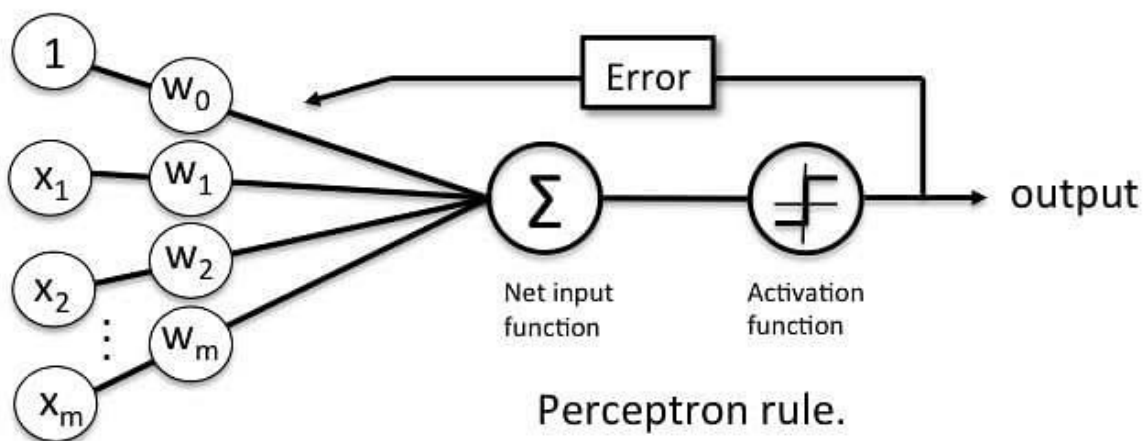
Perceptron rule.

The Perceptron receives multiple input signals, and if the sum of the input signals exceeds a certain threshold, it either outputs a signal or does not return an output. In the context of supervised learning and classification, this can then be used to predict the class of a sample.

Perceptron Function

Perceptron is a function that maps its input "x," which is multiplied with the learned weight coefficient; an output value "f(x)"is generated.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the equation given above:

- "w" = vector of real-valued weights
- "b" = bias (an element that adjusts the boundary away from origin without any dependence on the input value)
- "x" = vector of input x values

$$\sum_{i=1}^{m} w_i x_i$$

- "m" = number of inputs to the Perceptron

The output can be represented as "1" or "0." It can also be represented as "1" or "-1" depending on which activation function is used.

Output of Perceptron

Perceptron with a Boolean output:

Inputs: x1…xn

Output: o(x1….xn)

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Weights: wi=> contribution of input xi to the Perceptron output;

w0=> bias or threshold

If Σw.x > 0, output is +1, else −1. The neuron gets triggered only when weighted input reaches a certain threshold value.

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

$$sgn(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}$$

An output of +1 specifies that the neuron is triggered. An output of -1 specifies that the neuron did not get triggered.

"sgn" stands for sign function with output +1 or -1.

Lab Assignment No.	7
Title	Write a python program to show Back Propagation Network for XOR function with Binary Input and Output
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 7

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

Problem Statement: Implement to show Back Propagation Network for XOR function with Binary Input and Output

.

<mark>Objective:</mark>

- Understand and implement to show Back Propagation Network for XOR function with Binary Input and Output

<mark>Outcome:</mark>

- Ability to choose an appropriate problem-solving

Software Required:

- Python

Theory:

Artificial Neural Network (ANN) is a computational model based on the biological neural networks of animal brains. ANN is modeled with three types of layers: an input layer, hidden layers (one or more), and an output layer. Each layer comprises nodes (like biological neurons) are called Artificial Neurons. All nodes are connected with weighted edges (like synapses in biological brains) between two layers. Initially, with the forward propagation function, the output is predicted. Then through backpropagation, the weight and bias to the nodes are updated to minimizing the error

in prediction to attain the convergence of cost function in determining the final output.

XOR logical function truth table for *2-bit binary variables*, i.e, the input

vector   and the corresponding output   –

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Approach:
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

Conclusion: Thus we have Implement

Lab Assignment No.	08
Title	Write a python program to illustrate ART neural network
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 08

Title: Write a python program to illustrate ART neural network.

Problem Statement: Implement a python program to illustrate ART neural network.

Objective:

Outcome:

- Ability to choose an appropriate problem-solving method and knowledge representation technique

Software Required:

- Python

Theory:

Adaptive Resonance Theory (ART)

Adaptive resonance theory is a type of neural network technique developed by Stephen Grossberg and Gail Carpenter in 1987. The basic ART uses unsupervised learning techniques.

The term **"adaptive"** and **"resonance"** used in this suggests that they are open to new learning(i.e. adaptive) without discarding the previous or the old information(i.e. resonance).

The ART networks are known to solve the stability-plasticity dilemma i.e., stability refers to their nature of memorizing the learning and plasticity refers to the fact that they are flexible to gain new information.

Due to this nature of ART they are always able to learn new input patterns without forgetting the past. ART networks implement a clustering algorithm. Input is presented to the network and the algorithm checks whether it fits into one of the already stored clusters. If it fits then the input is added to the cluster that matches the most else a new cluster is formed.

Types of Adaptive Resonance Theory(ART)
Carpenter and Grossberg developed different ART architectures as a result of 20 years of research. The ARTs can be classified as follows:

- **ART1** – It is the simplest and the basic ART architecture. It is capable of clustering binary input values.
- **ART2** – It is an extension of ART1 that is capable of clustering continuous-valued input data.
- **Fuzzy ART** – It is the augmentation of fuzzy logic and ART.
- **ARTMAP** – It is a supervised form of ART learning where one ART learns based on the previous ART module. It is also known as predictive ART.
- **FARTMAP** – This is a supervised ART architecture with Fuzzy logic included.

Basic of Adaptive Resonance Theory (ART) Architecture
The adaptive resonant theory is a type of neural network that is self-organizing and competitive. It can be of both types, the unsupervised ones(ART1, ART2, ART3, etc) or the supervised ones(ARTMAP). Generally, the supervised algorithms are named with the suffix "MAP".
But the basic ART model is unsupervised in nature and consists of:

- F1 layer or the comparison field(where the inputs are processed)
- F2 layer or the recognition field (which consists of the clustering units)
- The Reset Module (that acts as a control mechanism)

The **F1 layer** accepts the inputs and performs some processing and transfers it to the F2 layer that best matches with the classification factor.
There exist **two sets of weighted interconnection** for controlling the degree of similarity between the units in the F1 and the F2 layer.

The **F2 layer** is a competitive layer. The cluster unit with the large net input becomes the candidate to learn the input pattern first and the rest F2 units are ignored.

The **reset unit** makes the decision whether or not the cluster unit is allowed to learn the input pattern depending on how similar its top-down weight vector is to the input vector and to the decision. This is called the vigilance test.

Thus we can say that the **vigilance parameter** helps to incorporate new memories or new information. Higher vigilance produces more detailed memories, lower vigilance produces more general memories.

Advantage of Adaptive Resonance Theory (ART)
- It exhibits stability and is not disturbed by a wide variety of inputs provided to its network.
- It can be integrated and used with various other techniques to give more good results.
- It can be used for various fields such as mobile robot control, face recognition, land cover classification, target recognition, medical diagnosis, signature verification, clustering web users, etc.
-
- It does not guarantee stability in forming clusters.

Limitations of Adaptive Resonance Theory

Some ART networks are inconsistent (like the Fuzzy ART and ART1) as they depend upon the order in which training data, or upon the learning rate.

Lab Assignment No.	09
Title	Write a python program to design a Hopfield Network which stores 4 vectors
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II

Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 09

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

Problem Statement: Design a Hopfield Network which stores 4 vectors

Objective:

Understand and Implement Hopfield Network which stores 4 vectors

Outcome:

Ability to choose a Hopfield Network which stores 4 vectors

Software Required:

● Python

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.

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 the following 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.
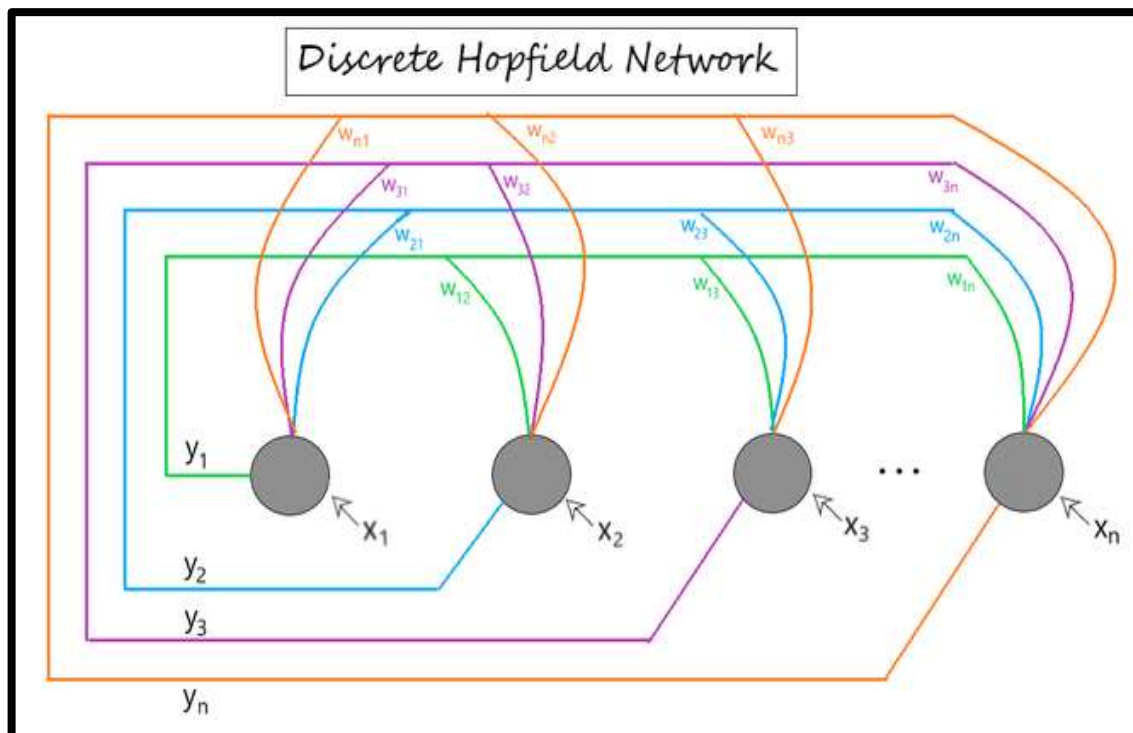
Conclusion: Thus we have

Lab Assignment No.	**10**
Title	Write Python program to implement CNN object detection. Discuss numerous performance evaluation metrics for evaluating the object-detecting algorithms' performance.
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory I

Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 10

Title: Write Python program to implement CNN object detection. Discuss numerous performance evaluation metrics for evaluating the object detecting algorithms' performance.

Problem Statement: Implement any one of the following Expert System

Objective:
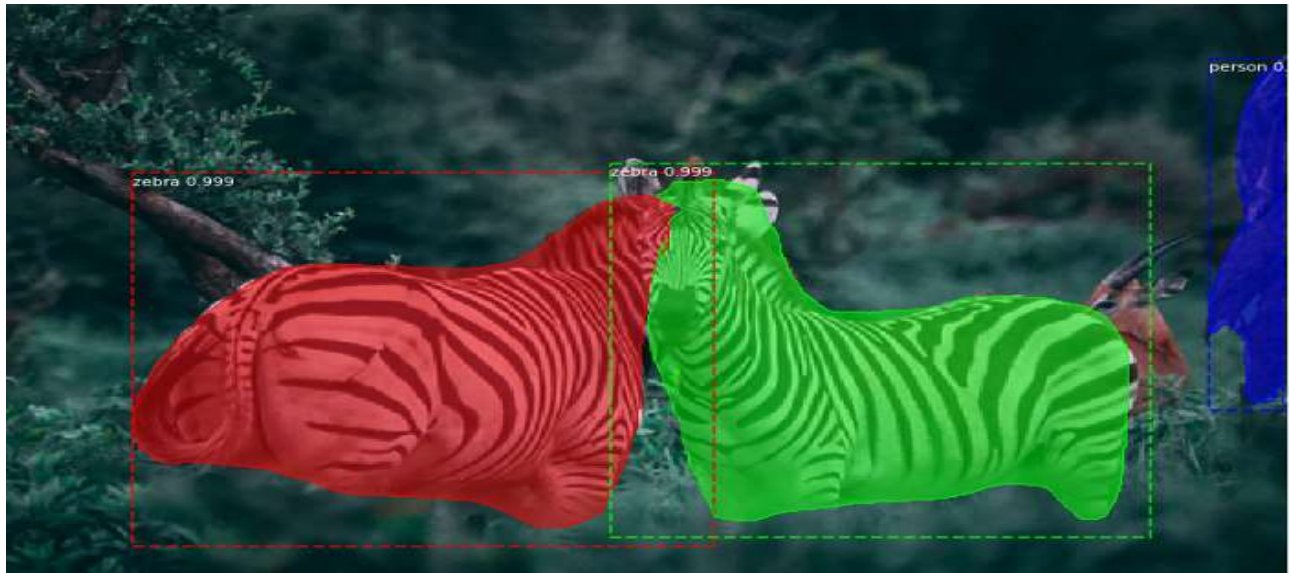
- Understand and implement

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory:

What is mAP? How to evaluate the performance of an object detection model?*What is mAP? How to calculate mAP along with 11-point interpolation?*

Object detection and instance segmentation

We use machine learning and deep learning to solve regression or classification problems.

We used Root Mean Square(RMS) or Mean Average Percentage Error(MAPE) etc. to evaluate the performance of a regression model.

Classification models are evaluated using Accuracy, Precision, Recall or an F1- Score.

Is object detection, classification or a regression problem?

Multiple deep learning algorithms exist for object detection like RCNNs: Fast RCNN, Faster RCNN, YOLO, Mask RCNN etc.

**The objective of an object detection model is to**

- **Classification**: Identify if an object is present in the image and the class of the object

- **Localization**: Predict the co-ordinates of the bounding box around the object when an object is present in the image. Here we compare the co-ordinates of ground truth and predicted bounding boxes

We need to evaluate the performance of both classification as well as localization of using bounding boxes in the image

How do we measure the performance of the object detection model?

For object detection, we use the concept of Intersection over Union (IoU). IoU computes the intersection over the union of the two bounding boxes; the bounding box for the ground truth and the predicted bounding box
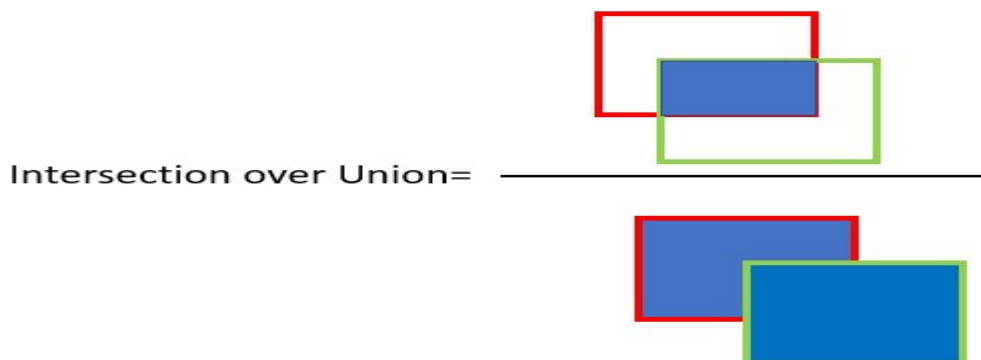
Red is the ground truth bounding box and green is the predicted bounding box

An IoU of 1 implies that the predicted and the ground-truth bounding boxes perfectly overlap.

You can set a threshold value for the IoU to determine if the object detection is valid or not.

Let's say you set IoU to 0.5, in that case,

- **if IoU ≥0.5,** classify the object detection as **True Positive(TP)**

- **if Iou <0.5**, then it is a false detection and classify it as **False Positive(FP)**

- **When ground truth is present in the image and model failed to detect the object,** classify it as **False Negative(FN).**

- **True Negative (TN)**: TN is every part of the image where we did not predict an object. This metric is not useful for object detection, hence we ignore TN.

Set the IoU threshold value to 0.5 or greater. It can be set to 0.5, 0.75. 0.9 or 0.95 etc.

Use Precision and Recall as the metrics to evaluate the performance. Precision and Recall are calculated using true positives(TP), false positives(FP) and false negatives(FN).

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

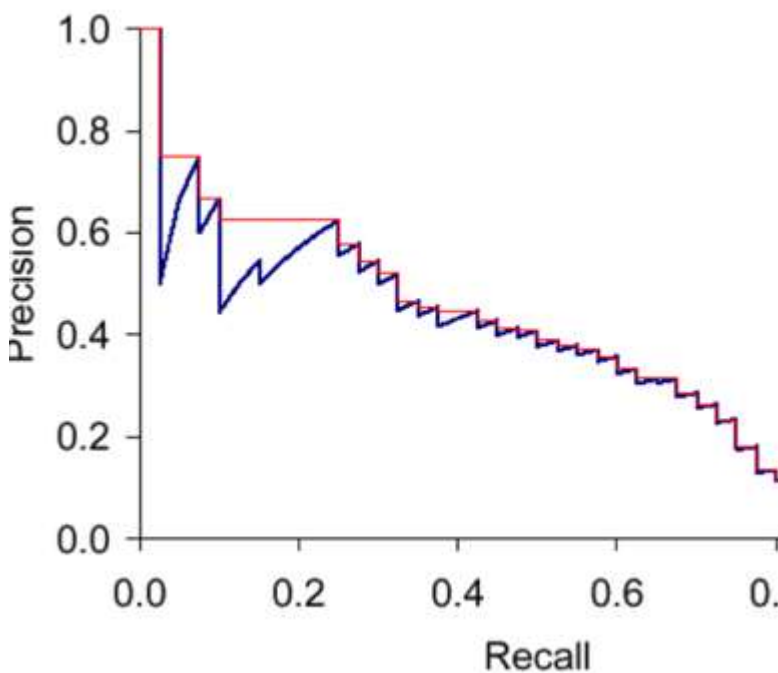Calculate precision and recall for all objects present in the image.

You also need to consider the confidence score for each object detected by the model in the image. Consider all of the predicted bounding boxes with a confidence score above a certain threshold. Bounding boxes above the threshold value are considered as positive boxes and all predicted bounding boxes below the threshold value are considered as negative.

use 11-point interpolated average precision to calculate mean Average Precision(mAP)

How to calculate mAP using 11 point interpolation?

Step 1: Plot Precision and Recall

Plot the precision and recall values on a Precision-Recall (PR) graph. PR graph is monotonically decreasing, there is always a trade-off between precision and recall. Increasing one will decrease the other. Sometimes PR graph is not always monotonically decreasing due to certain exceptions and/or lack of data.

Source: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html

Step 2: Calculate the mean Average Precision(mAP), use 11 point interpolation technique.

Interpolated precision is average precision measured at 11 equally spaced recall levels of 0.0, 0.1, 0.2, 0.3 ….0.9, 1.0 as shown in the figure above

The PR graph sometimes may not be monotonically decreasing, to resolve the issue, we set a max of precision for a value of recall. Graphically, at each recall level, we replace each precision value with the maximum precision value to the right of that recall level i.e.; we take the maximum of all future points

The rationale is the willingness to look at higher precision values if both precision and recall get better.

Finally, calculate the arithmetic mean of the interpolated precision at each recall level for each piece of information in the test collection.

the mAP is always calculated over the entire dataset.

Let's understand with an example as shown below, recall values are sorted for us to plot the PR graph

Recall	Precision
0.20	1.00
0.20	0.50
0.40	0.40
0.40	0.67
0.60	0.50
0.60	0.50
0.80	0.57
0.80	0.44
1.00	0.50
1.00	0.47

Sample Precision and Recall values

11 point interpolation will use the highest value for the precision for a recall value.

We create 11 equally spaced recall levels of 0.0, 0.1, 0.2, 0.3 ....0.9, 1.0

Recall of 0.2 has the highest precision value of 1.00. The recall value of 0.4 has different precision values of 0.4, 0.6and 7, 0.5. In this scenario, we use the highest precision value of 0.67. When the precision value is 0.6, we have a precision value of 0.5 but for a recall

of 0.8, we see a higher precision value of 0.57. Based on the rationale for 11-point interpolation, we take the maximum of all future points, so the precision that we need to consider is 0.57 instead of 0.5. Finally, for a recall of 1.0, we take the max precision which is 0.5.

Recall	Interpolated Precision
0.00	1.00
0.10	1.00
0.20	1.00
0.30	1.00
0.40	0.67
0.50	0.67
0.60	0.57
0.70	0.57
0.80	0.57
0.90	0.57
1.00	0.50
mAP	0.74

Now plotting the Precision-Recall as well as the Interpolated precision.

Precision Recall Graph
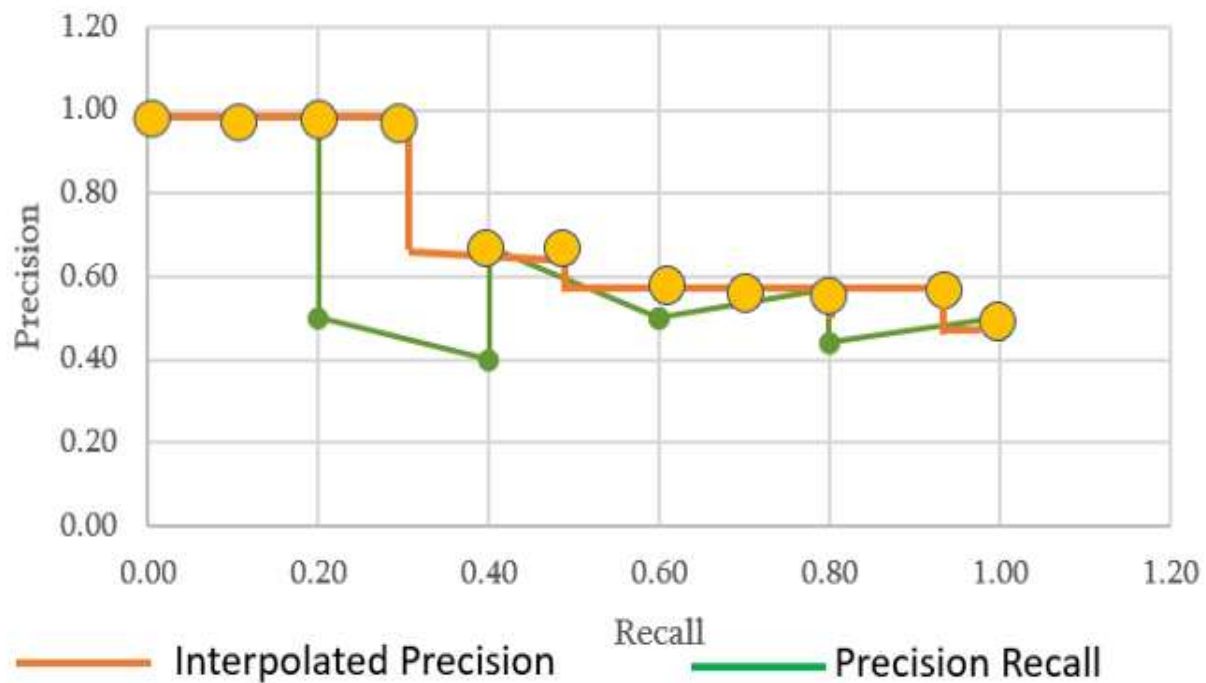
We finally apply the mean average precision formula

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{\text{interp}(r)}$$

AP = 1/11(4* 1.0 + 2 * 0.67+ 4*0.57 + 1*0.5) =0.74

Conclusion: Thus we have implemented

Group C

Lab Assignment No.	**11**
Title	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.
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 11

Title: 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.

Problem Statement: Implement an image classification challenge, create and train a ConvNet in Python using TensorFlow

Objective:

- Understand and implement an image classification challenge, create and train a ConvNet in Python using TensorFlow

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory: **TensorFlow is a popular deep learning framework. In this tutorial, you will learn the basics of this Python library and understand how to implement these deep, feed-forward artificial neural networks with it.**

Tensors

In layman's terms, a tensor is a way of representing the data in deep learning. A tensor can be a 1-dimensional, a 2-dimensional, a 3-dimensional array, etc. You can think of a tensor as a multidimensional array. In machine learning and deep learning, you have datasets that are high dimensional, in which each dimension represents a different feature of that dataset.

These features are incorporated by the tensors.
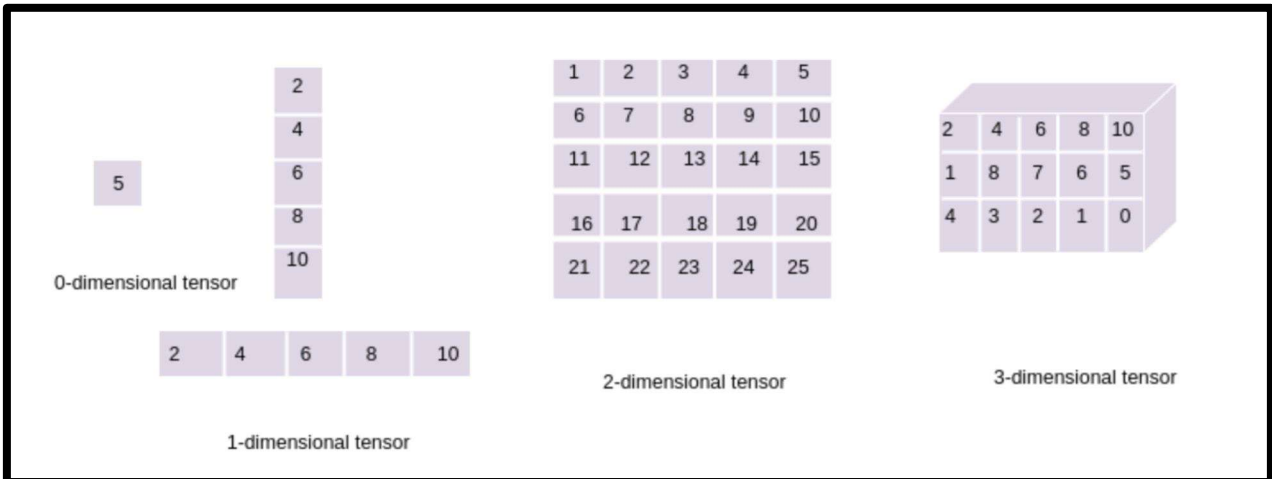
Tensors versus Matrices: Differences

A matrix is a two-dimensional grid of size $n×m$ that contains numbers: you can add and subtract matrices of the same size, multiply one matrix with another as long as the sizes are compatible $((n×m)×(m×p)=n×p)$, and multiply an entire matrix by a constant.

A vector is a matrix with just one row or column (but see below).

A tensor is often thought of as a generalized matrix. That is, it could be

- a 1-D matrix, like a vector, which is actually such a tensor,
- a 3-D matrix (something like a cube of numbers),
- a 0-D matrix (a single number), or
- a higher dimensional structure that is harder to visualize.

Convolutional Neural Network (CNN) in TensorFlow

Fashion-MNIST is similar to the MNIST dataset that you might already know, which you use to classify handwritten digits. That means that the image dimensions, training, and test splits are similar.

CIFAR-10 Photo Classification Dataset

CIFAR is an acronym that stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute.

The dataset is comprised of 60,000 32×32 pixel colour photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

- 0: airplane
- 1: automobile
- 2: bird
- 3: cat
- 4: deer
- 5: dog
- 6: frog
- 7: horse
- 8: ship
- 9: truck
  These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

CIFAR-10 is a well-understood dataset and widely used for benchmarking computer vision algorithms in the field of machine learning. The problem is "*solved*." It is relatively straightforward to achieve 80% classification accuracy. Top performance on the problem is achieved by deep learning convolutional neural networks with a classification accuracy above 90% on the test dataset

Lab Assignment No.	**12**
Title	TensorFlow/Pytorch implementation of CNN
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	
Assessor's Sign	

ASSIGNMENT No: 12

Title: TensorFlow/Pytorch implementation of CNN

Problem Statement: Implement TensorFlow/Pytorch implementation of CNN

Objective:

- Understand and implement TensorFlow/Pytorch of CNN

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

Theory: PyTorch is a Python framework for deep learning that makes it easy to perform research projects, leveraging CPU or GPU hardware. The basic logical unit in PyTorch is a tensor, a multidimensional array. PyTorch combines large numbers of tensors into computational graphs, and uses them to construct, train and run neural network architectures. A unique feature of PyTorch is that graphs are dynamic, written directly in Python, and can be modified during runtime.

Convolutional Neural Networks (CNN) are the basic architecture used in deep learning for computer vision. The Torch. nn library provides built-in functions that can create all the building blocks of CNN architectures:

- Convolution layers
- Pooling layers
- Padding layers
- Activation functions
- Loss functions
- Fully connected layers

How Do CNNs Work?

- A convolutional neural network (CNN for short) is a special type of neural network model primarily designed to process 2D image data, but which can also be used with 1D and 3D data.
- At the core of a convolutional neural network are two or more convolutional layers, which perform a mathematical operation called a "convolution". The convolution multiplies a set of weights with the inputs of the neural network. However, unlike in a regular neural network, this multiplication happens using a "window" that passes over the image, called a filter or kernel. As the filter passes over the image, each time the weights are multiplied by a specific set of input values.
- The mathematical operation performed during the convolution operation is a "dot product". This is an element-wise multiplication between the weights in the filter and the input values. The total is summed, giving a single value for each filter position. This operation is also called a "scalar product".
- Because the filter is usually smaller than the image used as an input, the same weights can be applied to the input multiple times. Specifically, the system applies the filter from right to left and from top to bottom to cover the entire image, with the objective of discovering important features in the image.
- It is a powerful idea to constantly apply the same filter to the whole image. If the filter can identify certain features in the image, it reviews the entire image and looks for that feature everywhere. This is called translation invariance—the CNN architecture is mainly interested in the presence of a feature, rather than its specific location.
- The values obtained from the convolution operation for each filter position (one value for each filter position) create a two-dimensional matrix of output values, which represent the features extracted from the underlying image. This output matrix is called a "feature map".
- Once the feature map is ready, any value in the functional map can be transmitted nonlinearly to the next convolutional layer (for example, via ReLU activation). The output of the convolutional layers sequence is transmitted to fully connected layers, which produce the final prediction, typically regarding a label describing the image.
-
- Creating a Validation Set

1. Implementing CNNs Using PyTorch

2. Generating Predictions for the Test Set

Lab Assignment No.	13
Title	MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow
Roll No.	
Class	TE
Date of Completion	
Subject	Software Laboratory II
Assessment Marks	

Assessor's Sign	

Title: MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Problem Statement: Implement MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Objective:

- Understand and implement MNIST Handwritten Character Detection using
- PyTorch,
- Keras and
- Tensorflow

Outcome:

- Ability to choose an appropriate problem solving method and knowledge representation technique

Software Required:

- Python

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 practising 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.

Human Visual System is a marvel of the world. People can readily recognise digits. But it is not as simple as it looks. The human brain has a million neurons and billions of connections between them, which makes this exceptionally complex task of image processing easier. People can effortlessly recognize digits.

However, it turns into a challenging task for computers to recognize digits. Simple hunches about how to recognize digits become difficult to express algorithmically. Moreover, there is a significant variation in writing from person to person, which makes it immensely complex.

Handwritten digit recognition system is the working of a machine to train itself so that it can recognize digits from different sources like emails, bank cheque, papers, images, etc.

Google Colab

Google Colab has been used to implement the network. It is a free cloud service that can be used to develop deep learning applications using popular libraries such as Keras, TensorFlow, PyTorch, and OpenCV. The most important feature that distinguishes Colab from other free cloud services is; it provides GPU and is totally free. Thus, if the PC is incompatible with hardware requirements or does not support GPU, then it is the best option because a stable internet connection is the only requirement.

MNIST Datasets

MNIST stands for "Modified National Institute of Standards and Technology". It is a dataset of 70,000 handwritten images. Each image is of 28x28 pixels i.e. about 784 features. Each feature represents only one pixel's intensity i.e. **from 0(white) to 255(black).** This database is further divided into 60,000 training and 10,000 testing images.

Phases of Implementation
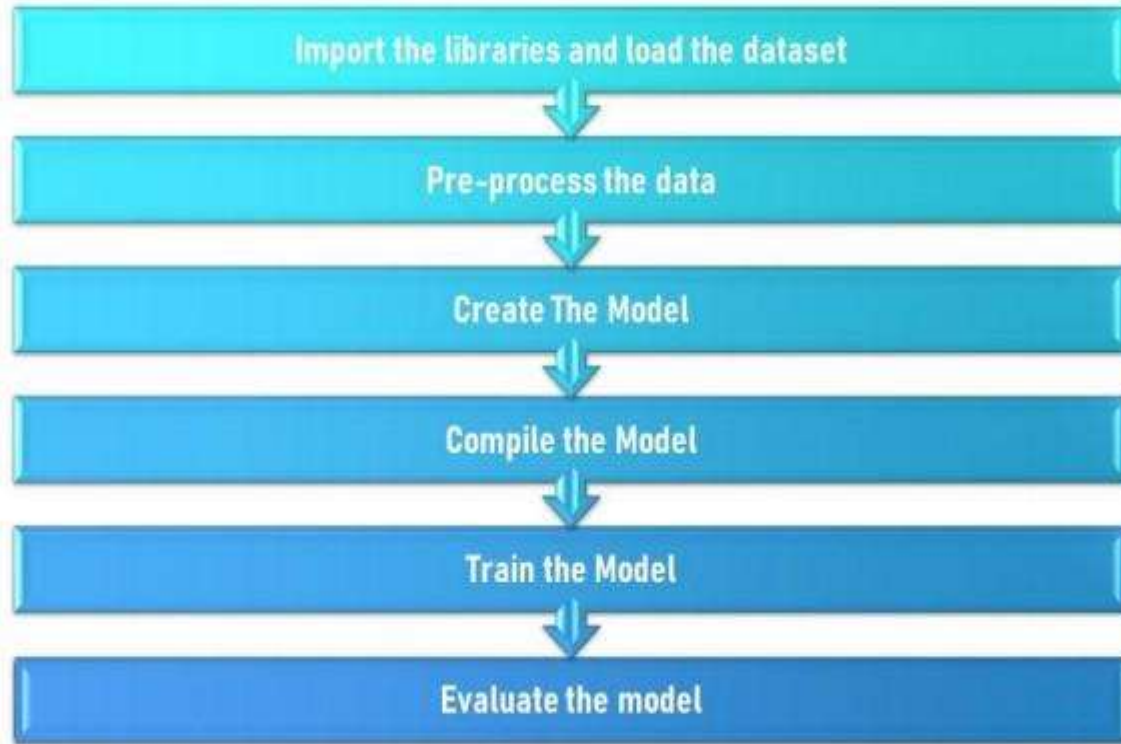

MNIST Handwritten Digit Classification Dataset

The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset.

It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9.

The task is to classify a given image of a handwritten digit into one of 10 classes representing 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 achieve a classification accuracy of above 99%, with an error rate between 0.4 %and 0.2% on the hold out test dataset.
The example below loads the MNIST dataset using the Keras API and creates a plot of the first nine images in the training dataset.
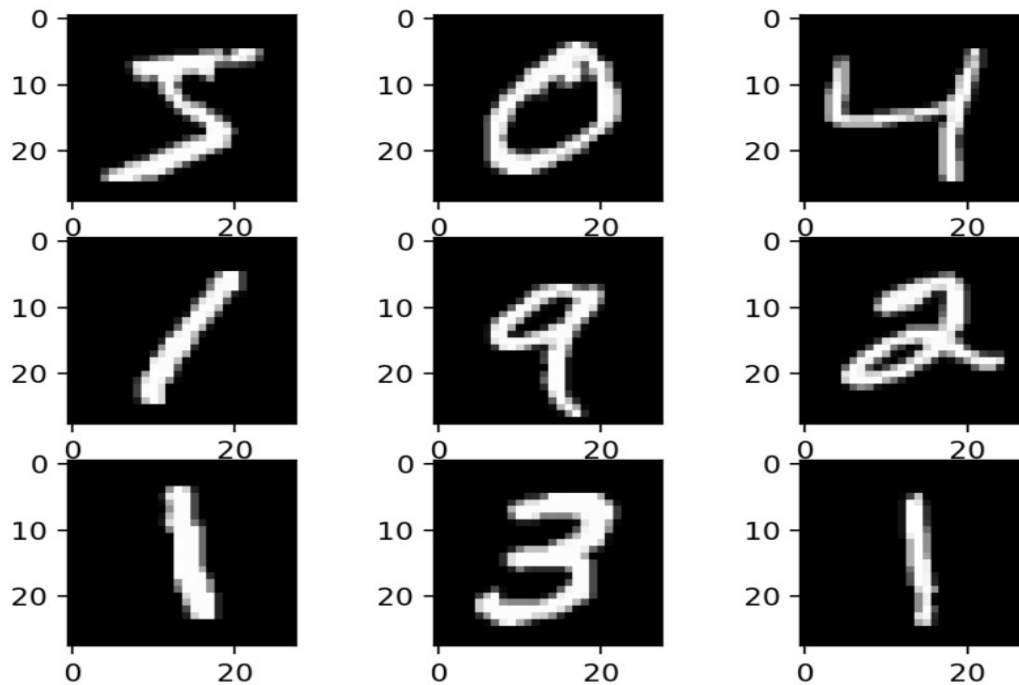
Plot of a Subset of Images From the MNIST Dataset

Model Evaluation Methodology

Although the MNIST dataset is effectively solved, it can be a useful starting point for developing and practising 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, which will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.