D Y Patil College of Engineering, Akurdi, Pune

Department of AI & DS Engineering

2022-23

Lab Manual

Software Laboratory II

(317523)

Artificial Neural Network

TE-SEM-II

Prepared By: Prof. Pranjali S. Bahalkar

INSTRUCTIONS TO STUDENTS

- The student should carry Identity card issued by the college.
- Student must sign in the log book provided when attending the lab session without fail.
- Come to the laboratory in time.
- Foods, drinks are NOT allowed.
- All bags must be kept at the indicated place.
- Read the Manual carefully before coming to the laboratory.
- Programs for the given problem statements specified in the lab manual should be logically correct with desired output.
- Write program and output after executing the code or attach printout of program and output to the file as a part of submission.
- Do check your lab assignment on the specified submission date given by subject teacher.

Sr. No.	Contents	Page No.
1.	Course Structure & Syllabus	4
2.	List of Lab Assignments	5
3.	Course Outcomes, Lab Outcomes	7
4.	Lab Assignment Plan	8
5.	Lab Assignment No. A1	10
6.	Lab Assignment No. A2	12
7.	Lab Assignment No. A3	15
8.	Lab Assignment No. A4	18
9.	Lab Assignment No. A5	20
10.	Lab Assignment No. A6	22
11.	Lab Assignment No. A7	25
12.	Lab Assignment No. A8	27
13.	Lab Assignment No. B1	29
14.	Lab Assignment No. B2	34
15.	Lab Assignment No. B3	37
16.	Lab Assignment No. B4	40
17.	Lab Assignment No. C1	44
18.	Lab Assignment No. C2	47
19.	Lab Assignment No. C3	49

Course Structure

Savitribai Phule Pune University

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

(With effect from Academic Year 2022-23)

Semester-VI

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

List of Lab 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 recognizeall 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 performanceevaluation 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. Alsotry to improve the performance of the model by applying various hyper parameter tuning to reduce the overfitting or under fitting problem that might occur. Maintain graphs of comparisons.

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

Mini Project

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

OR

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

Learning Resources

Text Books:

1.Neural Networks a Comprehensive Foundations, Simon Haykin, PHI edition.
2.Laurene Fausett:Fundamentals of Neural Networks: Architectures, Algorithms & Apps,Pearson, 2004.
3.Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python1st ed. Edition, Apress publication

Reference Books:

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

e-Books:

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

Course Outcomes, Lab Outcomes

COURSE OBJECTIVES:

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

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

COURSE OBJECTIVES:

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

CO1: Model artificial Neural Network, and to analyse 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.

CO-PO MAPPING

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

GROUP A

Assignment -1

Title: Activation functions that are being used in neural networks.

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 :
These are computational models and inspire by the human brain. Many of the recent advancements have been made in the field of Artificial Intelligence, including Voice Recognition, Image Recognition, Robotics using it. They are the biologically inspired simulations performed on the computer to perform certain specific tasks like -

- Clustering
- Classification
- Pattern Recognition

In general - It is a biologically inspired network of artificial neurons configured to perform specific tasks. These biological methods of computing are known as the next major advancement in the Computing Industry.
Linear Activation Function: The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), 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 it was given.

$$Linear$$

$$f(x) = x$$

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, as shown below.

$$Sigmoid\,/\,Logistic$$

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

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.

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

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.

$$f(x) = \max(0, x)$$

CONCLUSION:

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

Assignment- 2

Title: AND NOT function using McCulloch-Pitts Neural Net

Aim: Write a MATLAB program to 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 these 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.

$$f\left(\sum_{i=1}^{n} x_i w_i\right)$$

Activation Function:

Output Function ANDNOT Function:

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

Truth Table:

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

A NN with two input neurons and a single output neuron can operate as an ANDNOT logic function if we choose weights

W1 = 1, W2 = -1 and threshold Θ = 1.

Yin is a activation value
X1=1, X2=1,
Yin= W1*X1+W2*X2=1*1+(-1)*1=0, Yin< Θ , so Y=0
X1=1, X2=0
Yin=1*1+0*(-1)=1, Yin= Θ , so Y=1
X1=0, X=1
Yin=0-1=-1, Yin< Θ , so Y=0
X1=0, X2=0
Yin=0, Yin< Θ , so Y=0
So , Y=[0 1 0 0]

EXPECTED OUTPUT / CALCULATION / RESULT:

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

Output:

w1=1
w2=-1
Threshold:
Theta=1

With Output of Neuron:
0 1 0 0

Conclusion:

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

Assignment- 3

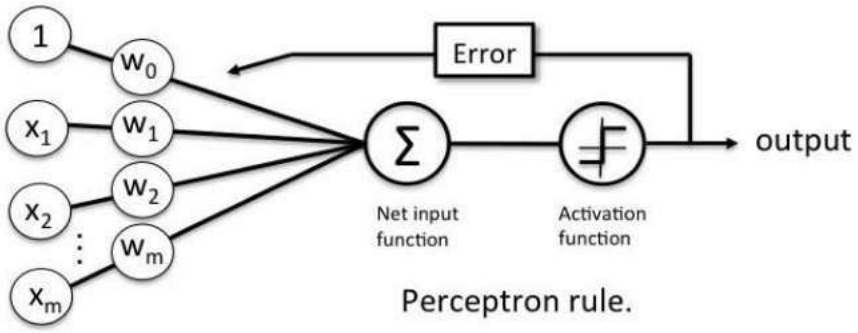
Title: Perceptron Neural Network to recognise even and odd numbers. Given numbers are in ASCII form 0 to 9

Aim: Program using Perceptron Neural Network to recognise even and odd numbers. Given numbers are in ASCII form 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 perceptrons laid out in two or more layers. Input to different perceptrons in a particular layer will be fed from the previous layer by combining them with different weights.



Perceptron rule.

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:

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

b = b + alpha * (y - y_hat)

Return the learned weight vector w and bias term b.

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.

Assignment- 4

Title: Demonstrate the Perceptron learning law with its decision regions.

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.

$$Sum = \sum_{i=1}^{N} I_i W_i + b,$$

where b represents the bias value.

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 < 1.0 4.

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

Repeat steps 2 and 3 until: 34 Department of Computer Engineering :

1. the iteration error is less than a user-specified error threshold.
2. a predetermined number of iterations have been completed.

Conclusion:

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

Assignment – 5

Title:

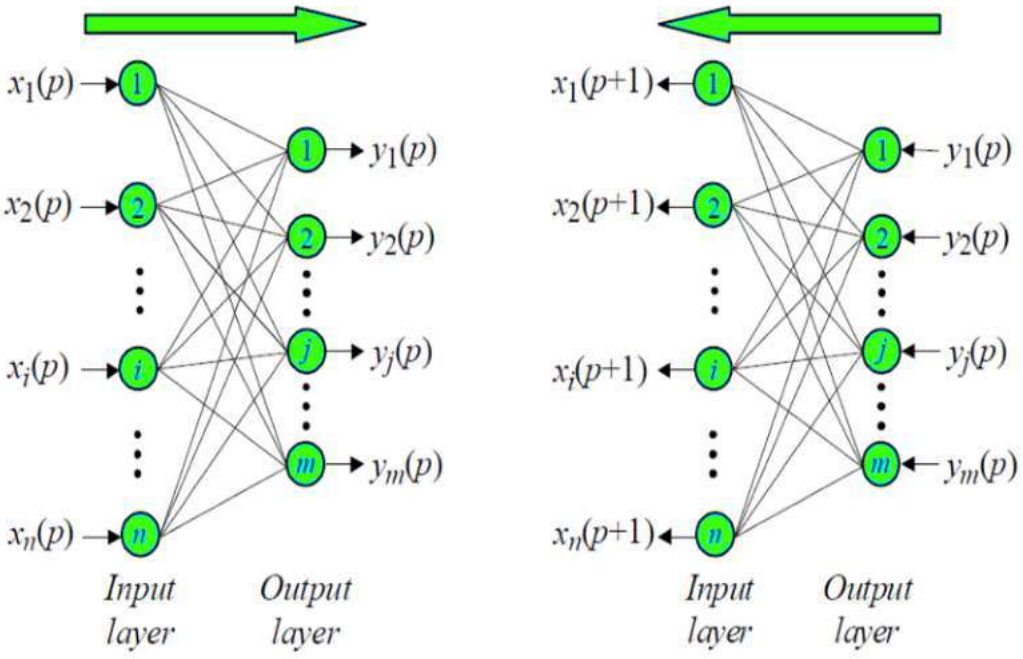
Bidirectional Associative Memory with two pairs of vectors.

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

Objective: To learn about Bidirectional Associative Memory with two pairs of vectors.

Theory:

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.



(*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.

Conclusion:

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

Assignment-6

Title:

Recognize the number 0, 1, 2, 39. A 5 * 3 matrices form 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.

Aim:

To recognize the number 0, 1, 2, 39. A 5 * 3 matrices form the numbers. For any valid point it is taken as 1 and invalid point it is taken as 0.

Objective:

To learn Digit Recognition in Python

Theory:

The described problem is a simple image classification problem, where we need to recognize a digit based on its pixel values in a 5x3 matrix. To solve this problem, we can use a neural network, which is a powerful machine learning model that is well-suited for image classification tasks.

Steps involved in building and training a neural network:-

1. Prepare a dataset of input-output pairs for training and testing the network.
2. Define a neural network architecture that can take a 5x3 matrix as input and output one of the four digits.
3. Train the network using the input-output pairs and an optimization algorithm.
4. Test the network on a set of input-output pairs that it has not seen before to evaluate its performance.
5. Experiment with different hyperparameter values to optimize the network's performance.
6. Use the trained network to predict the digit for any new 5x3 matrix by feeding it into the network and looking at the output node with the highest value.

Conclusion: We have successfully implemented to recognize the number 0, 1, 2, 39. A 5 * 3 matrices form the numbers. For any valid point it is taken as 1 and invalid point it is taken as 0.

Assignment-7

Title: Artificial Neural Network training process of Forward Propagation, Back Propagation.

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.

Algorithm:

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 l in the network, compute the weighted input z_l for that layer:

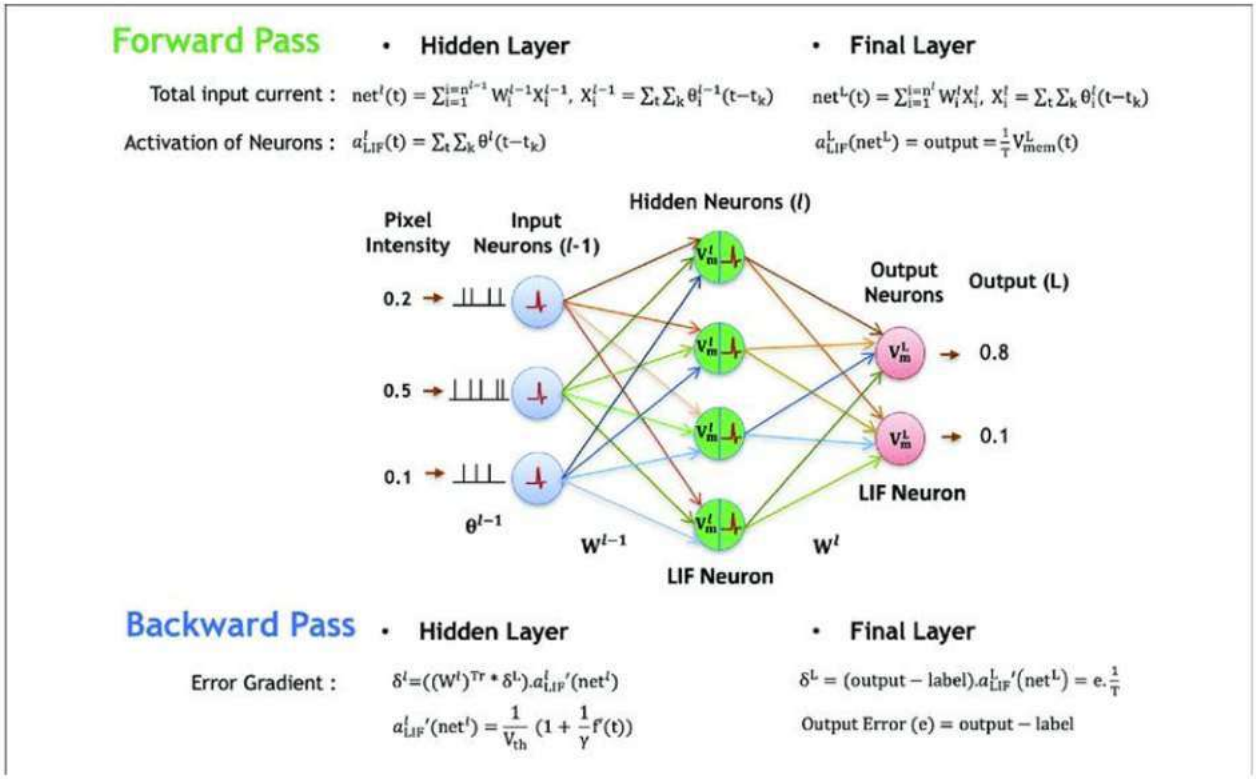
z = w * a-1 + b

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.



Conclusion:

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

Assignment-8

Title: Neural network architecture to do multi-class classification on any data.

Aim: Create a Neural network architecture from scratch in Python and use it to do multi-class classification on any data.

Objective: To learn about Neural network architecture to do multi-class classification on any data.

Theory:

What is Neural Network Architecture?

The architecture of neural networks is made up of an input, output, and hidden layer. Neural networks themselves, or artificial neural networks (ANNs), are a subset of machine learning designed to mimic the processing power of a human brain. Neural networks function by passing data through the layers of an artificial neuron.

Main Components of Neural Network Architecture:

There are many components to a neural network architecture. Each neural network has a few components in common:

Input - Input is data that is put into the model for learning and training purposes.

Weight - Weight helps organize the variables by importance and impact of contribution.

Transfer function - Transfer function is when all the inputs are summarized and combined into one output variable.

Activation function - The role of the activation function is to decide whether or not a specific neuron should be activated. This decision is based on whether or not the neuron's input will be important to the prediction process.

Bias - Bias shifts the value given by the activation function.

Algorithm:

Inputs:

X: input features

Y: target class labels

n_l: number of neurons in layer l

L: number of layers in the network

alpha: learning rate

num_iterations: number of iterations to run gradient descent

activation: activation function to use for hidden layers

output_activation: activation function to use for output layer

mini_batch_size: size of mini-batch to use in mini-batch gradient descent

print_cost: whether or not to print the cost during training

Outputs:

parameters: a dictionary of learned parameters

Steps:

Initialize the parameters for the neural network:

Use random initialization for weight matrices W^[l] and bias vectors b^[l] for all layers l in range(1, L)

Perform mini-batch gradient descent to train the network:

For each iteration of gradient descent:

Randomly shuffle the training examples X and corresponding labels Y

Split the data into mini-batches of size mini_batch_size

For each mini-batch:

Perform forward propagation to compute the activations for each layer of the network

Compute the cost using the output activations and the true labels Y

Use backpropagation to compute the gradients of the cost with respect to the parameters for each layer of the network

Update the parameters using the gradients and the learning rate alpha

Optionally, print the cost after each iteration

Return the learned

Conclusion:

We have successfully implemented Neural network architecture from scratch in Python and use it to do multi-class classification on any data.

GROUP B

ANN Lab Manual

Assignment B-1

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

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

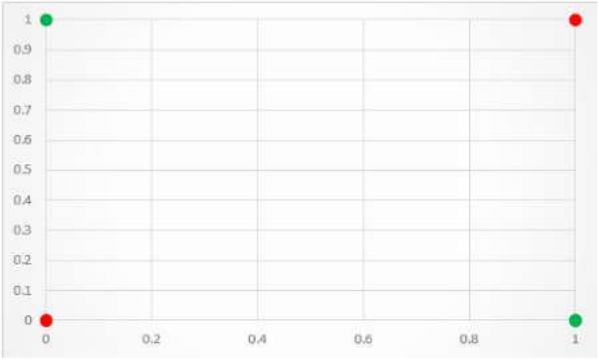
Theory:

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

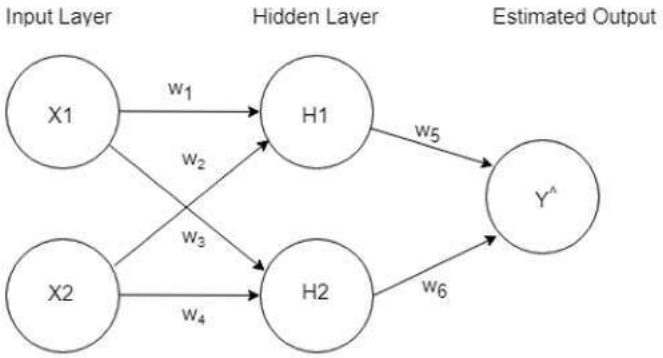
On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s, known as perceptrons. A limitation of this architecture is that it is only capable of separating data points with a single line. 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.

Model



Inputs

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

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

At hidden layer:

$$H_1 = x_1 w_1 + x_2 w_2$$

$$H_2 = x_1 w_3 + x_2 w_4$$

At output layer:

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

here, σ represents sigmoid function.

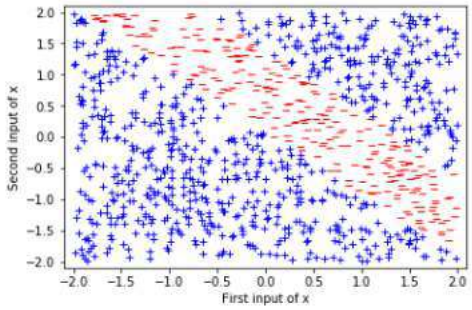
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Loss function:

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

Results

Classification WITHOUT gaussian noise



(731, 269)

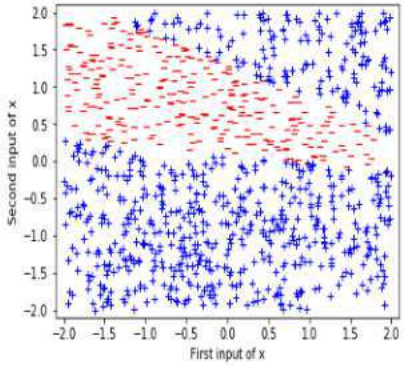
Observation : To classify, I generated 1000 x 2 random floats in range -2 to 2. Using weights from trained model, I classified each input & plotted it on 2-D space. Out of 1000, 731 points were classified as "+1" and 269 points were classified as "-1" It is clearly seen, classification region is not a single line, rather the 2-D region is separated by "-1" class. I did the same for classifications with Gaussian noise.

Classification WITH gaussian noise

In real applications, we almost never work with data without noise. Now instead of using the above points generate Gaussian random noise centered on these locations.

$$x_1 \sim \mu_1 = [1,1]^T, \Sigma_1 = \Sigma \quad y_1 = +1$$
$$x_2 \sim \mu_1 = [0,0]^T, \Sigma_2 = \Sigma \quad y_1 = +1$$
$$x_3 \sim \mu_1 = [1,0]^T, \Sigma_3 = \Sigma \quad y_1 = -1$$
$$x_4 \sim \mu_1 = [0,1]^T, \Sigma_4 = \Sigma \quad y_1 = -1$$
$$\Sigma = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix}$$

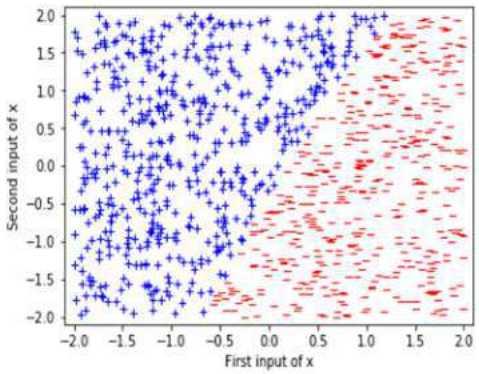
σ = 0.5



(702, 298)

Observation : We see a shift in classification regions. Here, classification is still not separated by a line

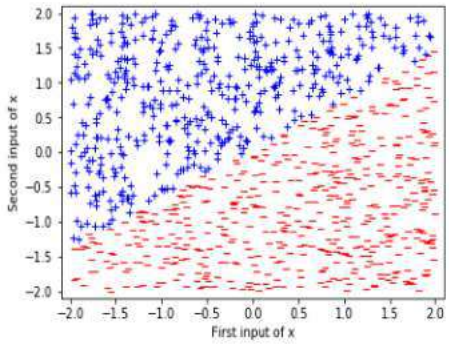
σ = 1.0



(538, 462)

Observation : We observe classifications being divided into two distinct regions.

σ = 2.0



(467, 533)

Observation : We see a shift in classification regions (compared to of $\sigma = 1$). Here also, we observe two distinct regions of classification.

Conclusion:

We have successfully implemented XOR problem with Back Propagation Algorithm.

ANN Lab Manual

Assignment No. B-3

Title: program for creating a back propagation feed-forward neural network.

Aim: Write a python program for creating a back propagation feed-forward neural network

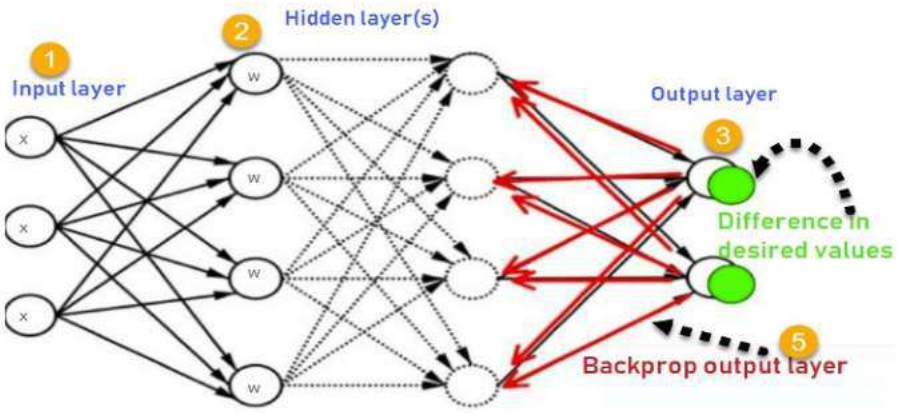
Theory:

Backpropagation Neural Network:

Backpropagation is a widely used algorithm for training feedforward neural networks. It computes the gradient of the loss function with respect to the network weights. It is very efficient, rather than naively directly computing the gradient concerning each weight. This efficiency makes it possible to use gradient methods to train multi-layer networks and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are often used.

The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight via the chain rule, computing the gradient layer by layer, and iterating backward from the last layer to avoid redundant computation of intermediate terms in the chain rule.

Backpropagation Algorithm:



Step 1: Inputs X, arrive through the preconnected path.

Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.

Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.

Step 4: Calculate the error in the outputs

Backpropagation Error= Actual Output – Desired Output

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.

Step 6: Repeat the process until the desired output is achieved.

Parameters :

x = inputs training vector x=(x1,x2,……xn).
t = target vector t=(t1,t2……tn).
δk = error at output unit.
δj = error at hidden layer.
α = learning rate.
V0j = bias of hidden unit j.

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the stepsstopping condition is to be false do step 3 to 10.

Step 3: For each training pair do step 4 to 9 (Feed-Forward).

Step 4: Each input unit receives the signal unit and transmits the signal xi signal to all the units.

Step 5 : Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input

$$zinj = v0j + \Sigma xivij \quad (i=1 \text{ to } n)$$

Applying activation function zj = f(zinj) and sends this signal to all units in the layer about i.e output units

For each output l=unit yk = (k=1 to m) sums its weighted input signals.

$$yink = w0k + \Sigma ziwjk \quad (j=1 \text{ to } a)$$

and applies its activation function to calculate the output signals.

$$yk = f(yink)$$

Backpropagation Error :

Step 6: Each output unit yk (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as:

$$\delta k = (tk - yk) + yink$$

Step 7: Each hidden unit Zj (j=1 to a) sums its input from all units in the layer above

$$\delta inj = \Sigma \delta j \, wjk$$

The error information term is calculated as :

$$\delta j = \delta inj + zinj$$

Updation of weight and bias :

Step 8: Each output unit yk (k=1 to m) updates its bias and weight (j=1 to a). The weight correction term is given by :

$$\Delta wjk = \alpha \, \delta k \, zj$$

and the bias correction term is given by $\Delta wk = \alpha \, \delta k$.

therefore $wjk(new) = wjk(old) + \Delta wjk$

$$w0k(new) = wok(old) + \Delta wok$$

for each hidden unit zj (j=1 to a) update its bias and weights (i=0 to n) the weight connection term

$$\Delta vij = \alpha \, \delta j \, xi$$

and the bias connection on term

$$\Delta v0j = \alpha \, \delta j$$

Therefore $vij(new) = vij(old) + \Delta vij$

$$v0j(new) = v0j(old) + \Delta v0j$$

Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

Need for Backpropagation:

Backpropagation is "backpropagation of errors" and is very useful for training neural networks. It's fast, easy to implement, and simple. Backpropagation does not require any parameters to be set, except the number of inputs. Backpropagation is a flexible method because no prior knowledge of the network is required.

Conclusion:

We have successfully implemented back propagation feed-forward neural network.

Assignment No. B4

Title: Python program for creating a back propagation feed-forward neural network.

Aim: Write a python program for creating a back propagation feed-forward neural network.

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.

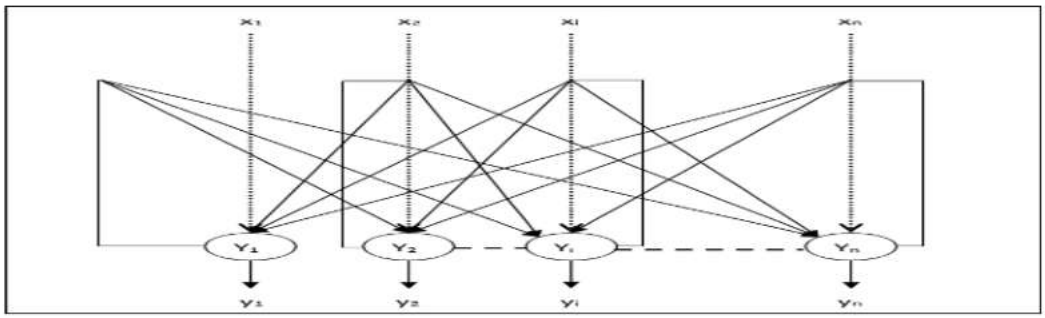
Architecture:

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 w1n 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 :

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

Here θi is the threshold.

Step 8 − Broadcast this output yi to all other units.

Step 9 − Test the network for conjunction.

Continuous Hopfield Network:

Unlike the discrete hopfield networks, here the time parameter is treated as a continuous variable. So, instead of getting binary/bipolar outputs, we can obtain values that lie between 0 and 1. It can be used to solve constrained optimization and associative memory problems. The output is defined as:

$$v_i = g(u_i)$$

where,

vi = output from the continuous hopfield network
ui = internal activity of a node in continuous hopfield network.

Conclusion:

We have successfully implemented back propagation feed-forward neural network.

Assignment No. B-5

Title:

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

Aim:

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

Theory:

Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. When humans look at images or video, we can recognize and locate objects of interest within a matter of moments. The goal of object detection is to replicate this intelligence using a computer. Average Precision (AP) and mean Average Precision (mAP) are the most popular metrics used to evaluate object detection models, such as Faster R_CNN, Mask R-CNN, and YOLO, among others.

Definition of terms:

- **True Positive (TP)** — Correct detection made by the model.
- **False Positive (FP)** — Incorrect detection made by the detector.
- **False Negative (FN)** — A Ground-truth missed (not detected) by the object detector.
- **True Negative (TN)** —This is the background region correctly not detected by the model. This metric is not used in object detection because such regions are not explicitly annotated when preparing the annotations.

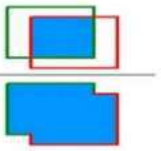
Object Detection metrics:

Intersection over Union (IoU):
IoU metric in object detection evaluates the degree of overlap between the ground(gt) truth and prediction(pd). The ground truth and the prediction can be of any shape-rectangular box, circle, or irregular shape). It is calculated as follows:

$$\text{IoU} = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)}$$

Diagrammatically, IoU is defined as follows (the area of the intersection divided by the area of union between ground-truth and predicted box.

$$IOU = \frac{\text{area of overlap}}{\text{area of union}} =$$



IoU ranges between 0 and 1, where 0 shows no overlap, and 1 means perfect overlap between gt and pd. IoU metric is useful through thresholding; that is, we need a threshold (α, say) to determine whether detection is correct.

Precision and Recall

Precision is the degree of exactness of the model in identifying only relevant objects. It is the ratio of TPs over all detections made by the model.

Recall measures the ability of the model to detect all ground truths— proposition of TPs among all ground truths.

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{TP}{\text{all detections}}$$
$$R = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{TP}{\text{all ground-truths}}$$

A model is said to be good if it has high precision and high recall. A perfect model has zero FNs and zero FPs (precision=1 and recall=1). Often, attaining a perfect model is not feasible.

Average Precision

AP@α is Area Under the Precision-Recall Curve(AUC-PR) evaluated at α IoU threshold. Formally, it is defined as follows.

$$AP@\alpha = \int_0^1 p(r)\, dr$$

Notation: AP@α or APα means that AP precision is evaluated at α IoU threshold. If you see metrics like AP50 and A75, they mean AP calculated at IoU=0.5 and IoU=0.75, respectively.

Mean Average Precision (mAP)

Remark (AP and the number of classes): AP is calculated individually for each class. This means that there are as many AP values as the number of classes (loosely). These AP values are averaged to obtain the **mean Average Precision (mAP)** metric.

Definition: The mean Average Precision (mAP) is the average of AP values over all classes.

$$\text{mAP}@\alpha = \frac{1}{n}\sum_{i=1}^{n} \text{AP}_i \quad \text{for n classes.}$$

Conclusion:

We have successfully implemented CNN object detection. Discuss numerous performance evaluation metrics for evaluating the object detecting algorithms' performance.

GROUP C

Assignment No C-1

Title:

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

Problem Statement:

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

Objective:

To Train a Neural Network with TensorFlow/Pytorch and evaluation of logistic regression using tensorFlow

Software Required:

TensorFlow/Pytorch

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 (AI) 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.

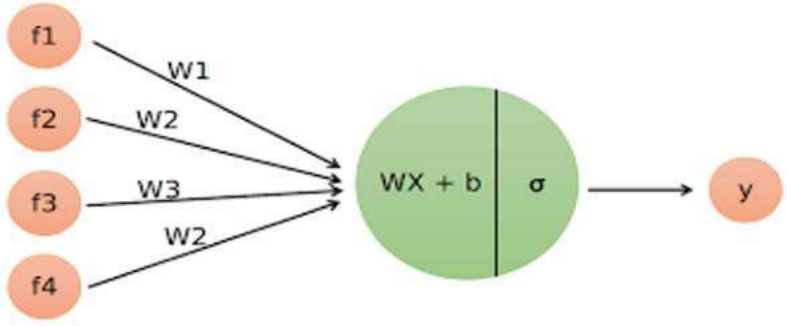
What is neural network?

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).

Diagram:



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

Conclusion:

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

Assignment No C-2

Title:

Implementation of CNN using Tensorflow/Pytorch

Problem Statement:

TensorFlow/Pytorch implementation of CNN

Objective:

To implement CNN using Tensorflow/Pytorch

Software Required:
Tensorflow/ Pytorch

Theory:

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.
How do convolutional neural networks work?

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- Convolutional layer
- Pooling layer
- Fully-connected (FC) layer

The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer. With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

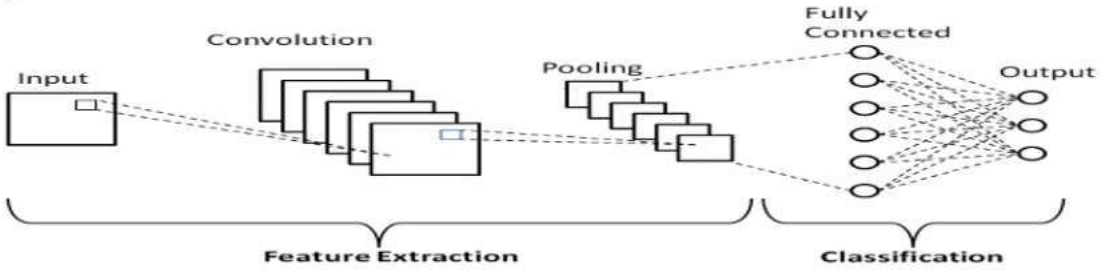
Advantages of Convolutional Neural Networks (CNNs):

- Good at detecting patterns and features in images, videos and audio signals.
- Robust to translation, rotation and scaling invariance.
- End-to-end training, no need for manual feature extraction.
- Can handle large amounts of data and achieve high accuracy.

Disadvantages of Convolutional Neural Networks (CNNs):

- Computationally expensive to train and require a lot of memory.
- Can be prone to overfitting if not enough data or proper regularization is used.
- Requires large amount of labeled data.
- Interpretability is limited, it's hard to understand what the network has learned.

Diagram:



Algorithm:

Step 1 – Include the necessary modules for TensorFlow and the data set modules, which are needed to compute the CNN model.

Step 2 – Declare a function called **run_cnn()**, which includes various parameters and optimization variables with declaration of data placeholders. These optimization variables will declare the training pattern.

Step 3 – In this step, we will declare the training data placeholders with input parameters . This is the flattened image data that is drawn from **mnist.train.nextbatch()**.

We can reshape the tensor according to our requirements.

Step 4 – Now it is important to create some convolutional layers –

Step 5 – Let us flatten the output ready for the fully connected output stage We can set up some weights and bias values for this layer, then activate with ReLU.

Step 6 – Another layer with specific softmax activations with the required optimizer defines the accuracy assessment, which makes the setup of initialization operator.

Step 7 – We should set up recording variables. This adds up a summary to store the accuracy of data.

Conclusion:

We have successfully implemented CNN using Tensorflow/Pytorch

ANN Lab Manual

Assignment No. 04

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

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

Objective:
To study and implement MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Software Required:
Tensorflow/ Pytorch

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, I 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 I have written digit 1 and it recognized it as 1 with 17% accuracy.

Conclusion:

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