



## LABORATORY MANUAL

### B.Tech. Semester- V

ARTIFICIAL NEURAL NETWORK LAB USING MATLAB

Subject code: LC-CSE-421G

**Prepared by:**

Prof Renu Narwal

**Checked by:**

Dr. Ashima Mehta

**Approved by:**

Prof. (Dr.) Isha Malhotra

**Sign.: .....**

**Sign.: .....**

**Sign.: .....**

## Table of Contents

1. Vision and Mission of the Institute
2. Vision and Mission of the Department
3. Programme Educational Objectives (PEOs)
4. Programme Outcomes (POs)
5. Programme Specific Outcomes (PSOs)
6. University Syllabus
7. Course Outcomes (COs)
8. CO- PO and CO-PSO mapping
9. Course Overview
10. List of Experiments
11. DOs and DON'Ts
12. General Safety Precautions
13. Guidelines for students for report preparation
14. Lab assessment criteria
15. Details of Conducted Experiments
16. Lab Experiments

## Vision and Mission of the Institute

### **Vision:**

“Empowering human values and advanced technical education to navigate and address global challenges with excellence.”

### **Mission:**

**M1:** Seamlessly integrate human values with advanced technical education.

**M2:** Supporting the cultivation of a new generation of innovators who are not only skilled but also ethically responsible.

**M3:** Inspire global citizens who are equipped to create positive and sustainable impact, driving progress towards a more inclusive and harmonious world.

## Vision and Mission of the Department

### **Vision:**

““Steering the future of computer science through innovative advancements, fostering ethical values and principles through technical education.”

”

### **Mission:**

**M1:** Directing future innovations in computer science through revolutionary progress.

**M2:** Instilling a foundation of ethical values and principles in every technologist.

**M3:** Offering a comprehensive technical education to equip individuals for a meaningful and influential future.

## **Programme Educational Objectives (PEOs)**

PEO1: Apply the technical competence in Computer Science and Engineering for solving problems in the real world.

PEO2: Carry out research and develop solutions on problems of social applications.

PEO3: Work in a corporate environment, demonstrating team skills, work morals, flexibility and lifelong learning.

---

## Programme Outcomes (POs)

**PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

**PO2: Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

## **Program Specific Outcomes (PSOs)**

PSO1: Exhibit design and programming skills to develop and mechanize business solutions using revolutionary technologies.

PSO2: Learn strong theoretical foundation leading to brilliance and enthusiasm towards research, to provide well-designed solutions to complicated problems.

PSO3: Work effectively with diverse Engineering fields as a team to design, build and develop system applications

---

## University Syllabus

1. To study about MATLAB.
2. Write a program to perform the basics matrix operations.
3. How the weight & bias value effects the output of neurons.
4. Write a program to plotting Activation Functions: Threshold functions, Signum function, Sigmoid function, Tan-hyperbolic function, Ramp function, Identity function using matlab
5. Write a Program to implement MP Model.
6. Write a program to generate ANDNOT function using McCulloch-Pitts neural net.
7. Write a program to generate XOR function using McCulloch-Pitts neural net
8. Write a Program to implement Hebb Model.
9. How the choice of activation function effect the output of neuron experiment with the following function purelin(n), binary threshold(hardlim(n) hardlims(n)) ,Tansig(n) logsig(n)
10. How the Perceptron Learning rule works for Linearly Separable Problem.
11. How the Perceptron Learning rule works for Non-Linearly Separable Problem.

## Course Outcomes (COs)

Upon successful completion of the course, the students will be able to:

C421.1: For a given conceptual problem student will be able to analyze the problem and able to visualize using NN

C422.2: Students will be familiar with different NN models and its implementation.

C423.3: Students will be able to understand the concept of learning in NN and its implementation.

C423.4: To conceptualize about perceptron learning rule works for linearly separable problems.

## CO-PO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C421.1	1	2	1	2	1	2			1			
C421.2	1	2	2	1		1	2	1				
C421.3	2	3	2	3	2	2	1		2	1		
C421.4	1	3	2	2		2	1		2	2		
C421.5	1	2	1	3	2	2	2	2		1		

## CO-PSO Mapping

	PSO1	PSO2	PSO3
C421.1	2	1	2
C421.2	2	3	1
C421.3	1	2	2
C421.4	2	1	1
C421.5	2	2	2

## Course Overview



The Artificial Neural Networks Lab is a practical course that focuses on implementing and experimenting with neural network models. This lab is designed to complement theoretical concepts covered in an artificial neural networks course and provides hands-on experience with neural network architectures, training algorithms, and applications. The lab begins with an introduction to artificial neural networks, their basic components, and the mathematical foundations. Students learn about different types of neural networks, such as feed forward, recurrent, and convolutional neural networks. Students study and implement various neural network architectures, including single-layer perceptron's, multi-layer perceptrons (MLPs), and deep neural networks (DNNs). They understand the structure, activation functions, and connections within these architectures. The lab covers different training algorithms used in neural networks, such as backpropagation, gradient descent, and stochastic gradient descent. Students learn how to train neural networks to minimize the error and optimize performance. The lab concludes with a project where students apply their knowledge and skills to solve a real-world problem using neural networks. They have the opportunity to design, implement, and evaluate a neural network model, considering data preprocessing, hyper parameter tuning, and performance analysis.

Throughout the lab, students are encouraged to experiment with different architectures, hyper parameters, and techniques to gain a deeper understanding of neural networks and their applications. They also learn best practices for model evaluation, interpretability, and troubleshooting common issues that may arise during implementation.

### **List of Experiments mapped with COs**

Si No.	Name of the Experiment	Course Outcome	Page No.
1	To study about MATLAB.	C421.2	1
2	Write a program to perform the basics matrix operations.	C421.2	8
3	How the weight & bias value effects the output of neurons.	C421.3	12
4	Write a program to plotting Activation Functions: Threshold functions, Signum function, Sigmoid function, Tan-hyperbolic function, Ramp function, Identity function using matlab	C421.3	14
5	Write a Program to implement MP Model.	C421.1	18
6	. Write a program to generate ANDNOT function using McCulloch-Pitts neural net.	C421.5	21
7	Write a program to generate XOR function using McCulloch-Pitts neural net	C421.3	24
8	Write a Program to implement Hebb Model.	C421.3	28
9	How the choice of activation function effect the output of neuron experiment with the following function purelin(n), bimary threshold(hardlim(n) haradlims(n)) ,Tansig(n) logsig(n)	C421.3	32
10	How the Perceptron Learning rule works for Linearly Separable Problem.	C421.4	36
11	How the Perceptron Learning rule works for Non-Linearly Separable Problem.	C421.2	40

## **DOs and DON'Ts**

### **DOs**

1. Login-on with your username and password.
2. Log off the computer every time when you leave the Lab.
3. Arrange your chair properly when you are leaving the lab.
4. Put your bags in the designated area.
5. Ask permission to print.

### **DON'Ts**

1. Do not share your username and password.
2. Do not remove or disconnect cables or hardware parts.
3. Do not personalize the computer setting.
4. Do not run programs that continue to execute after you log off.
5. Do not download or install any programs, games or music on computer in Lab.
6. Personal Internet use chat room for Instant Messaging (IM) and Sites is strictly prohibited.
7. No Internet gaming activities allowed.
8. Tea, Coffee, Water & Eatables are not allowed in the Computer Lab.

## **General Safety Precautions**

**Precautions (In case of Injury or Electric Shock)**

1. To break the victim with live electric source, use an insulator such as fire wood or plastic to break the contact. Do not touch the victim with bare hands to avoid the risk of electrifying yourself.
2. Unplug the risk of faulty equipment. If main circuit breaker is accessible, turn the circuit off.
3. If the victim is unconscious, start resuscitation immediately, use your hands to press the chest in and out to continue breathing function. Use mouth-to-mouth resuscitation if necessary.
4. Immediately call medical emergency and security. Remember! Time is critical; be best.

**Precautions (In case of Fire)**

1. Turn the equipment off. If power switch is not immediately accessible, take plug off.
2. If fire continues, try to curb the fire, if possible, by using the fire extinguisher or by covering it with a heavy cloth if possible isolate the burning equipment from the other surrounding equipment.
3. Sound the fire alarm by activating the nearest alarm switch located in the hallway.
4. Call security and emergency department immediately:

**Emergency : 200 (Reception)**

**Security : 248 (Gate No.1)**

## Guidelines to students for report preparation

All students are required to maintain a record of the experiments conducted by them. Guidelines for its preparation are as follows: -

- 1) All files must contain a title page followed by an index page. *The files will not be signed by the faculty without an entry in the index page.*
- 2) Student's Name, Roll number and date of conduction of experiment must be written on allpages.
- 3) For each experiment, the record must contain the following
  - (i) Aim/Objective of the experiment
  - (ii) Pre-experiment work (as given by the faculty)
  - (iii) Lab assignment questions and their solutions
  - (iv) Test Cases (if applicable to the course)
  - (v) Results/ output

### Note:

1. Students must bring their lab record along with them whenever they come for the lab.
2. Students must ensure that their lab record is regularly evaluated.

## Lab Assessment Criteria

An estimated 10 lab classes are conducted in a semester for each lab course. These lab classes are assessed continuously. Each lab experiment is evaluated based on 5 assessment criteria as shown in following table. Assessed performance in each experiment is used to compute CO attainment as well as internal marks in the lab course.

Grading Criteria	Exemplary (4)	Competent (3)	Needs Improvement (2)	Poor (1)
<b>AC1:</b> <b>Pre-Lab written work (this may be assessed through viva)</b>	Complete procedure with underlined concept is properly written	Underlined concept is written but procedure is incomplete	Not able to write concept and procedure	Underlined concept is not clearly understood
<b>AC2:</b> <b>Program Writing/ Modeling</b>	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/tools are applied, Program/solution written is readable	Assigned problem is properly analyzed, correct solution designed, appropriate language constructs/tools are applied	Assigned problem is properly analyzed & correct solution designed	Assigned problem is properly analyzed
<b>AC3:</b> <b>Identification &amp; Removal of errors/ bugs</b>	Able to identify errors/ bugs and remove them	Able to identify errors/ bugs and remove them with little bit of guidance	Is dependent totally on someone for identification of errors/ bugs and their removal	Unable to understand the reason for errors/ bugs even after they are explicitly pointed out
<b>AC4:</b> <b>Execution &amp; Demonstration</b>	All variants of input /output are tested, Solution is well demonstrated and implemented concept is clearly explained	All variants of input /output are not tested, However, solution is well demonstrated and implemented concept is clearly explained	Only few variants of input /output are tested, Solution is well demonstrated but implemented concept is not clearly explained	Solution is not well demonstrated and implemented concept is not clearly explained
<b>AC5:</b> <b>Lab Record Assessment</b>	All assigned problems are well recorded with objective, design constructs and solution along with Performance analysis using all variants of input and output	More than 70 % of the assigned problems are well recorded with objective, design constructs and solution along with Performance analysis is done with all variants of input and output	Less than 70 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis is done with all variants of input and output	Less than 40 % of the assigned problems are well recorded with objective, design contracts and solution along with Performance analysis done with all variants of input and output

# LAB EXPERIMENTS

# LAB EXPERIMENT 1

## **OBJECTIVE:**

To study about MATLAB.

## **BRIEF DESCRIPTION:**

Matlab is a high-level programming language and environment that is widely used in scientific and engineering applications. It stands for "Matrix Laboratory," and its primary focus is on numerical computations and data analysis. One of the key features of Matlab is its extensive built-in library of mathematical functions, which allows users to perform complex mathematical operations with ease. This library includes functions for linear algebra, statistics, optimization, signal processing, image processing, and more. Matlab also provides powerful tools for data visualization, allowing users to create plots, charts, and graphs to analyze and present their data effectively.

Overall, Matlab is a powerful tool for numerical computation, data analysis, and visualization. It is widely used in academia, industry, and research institutions for a wide range of applications, from developing algorithms and simulations to solving complex mathematical problems.

## **PRE-EXPERIMENT QUESTIONS:**

1. Which research field uses MATLAB for analysis?
2. Which tool is used for MATLAB?

## **Explanation:**

Developed primarily by Cleve Moler in the 1970's. Derived from FORTRAN subroutines LINPACK and EISPACK, linear and eigenvalue systems. Developed primarily as an interactive system to access LINPACK and EISPACK. Gained its popularity through word of mouth, because it was not socially distributed. Rewritten in C in the 1980's with more functionality, which include plotting routines.

The MathWorks Inc. was created (1984) to market and continue development of MATLAB. According to Cleve Moler, three other men played important roles in the origins of MATLAB: J. H. Wilkinson, George Forsythe, and John Todd. It is also interesting to mention the authors of LINPACK: Jack Dongara, Pete Steward, Jim Bunch, and Cleve Moler. Since then another package emerged: LAPACK. LAPACK stands for Linear Algebra Package. It has been designed to supersede LINPACK and EISPACK.

## **Introduction**

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Using the MATLAB



product, you can solve technical computing problems faster than with traditional programming languages, such as C, C++, and FORTRAN.

The name MATLAB stands for Matrix Laboratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research.

MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.

It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

Areas where MATLAB can be used

You can use MATLAB in a wide range of applications, including:

- Signal and image processing
- Communications
- control design
- Test and measurement
- Financial modeling and analysis
- and computational biology.

Add-on toolboxes (collections of special-purpose MATLAB functions, available separately) extend the MATLAB environment to solve particular classes of problems in these application areas.

MATLAB provides a number of features for documenting and sharing your work. You can integrate your MATLAB code with other languages and applications, and distribute your MATLAB algorithms and applications.

The MATLAB system consists of these main parts:

## **Desktop Tools and Development Environment**

This part of MATLAB is the set of tools and facilities that help you use and become more productive with MATLAB functions and files. Many of these tools are graphical user interfaces. It includes: the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, and browsers for viewing help, the workspace, and folders.

## **Mathematical Function Library**

This library is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

## **The Language**

The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick programs you do not intend to reuse. You can also do "programming in the large" to create complex application programs intended for reuse.

## **Graphics**

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

## **External Interfaces**

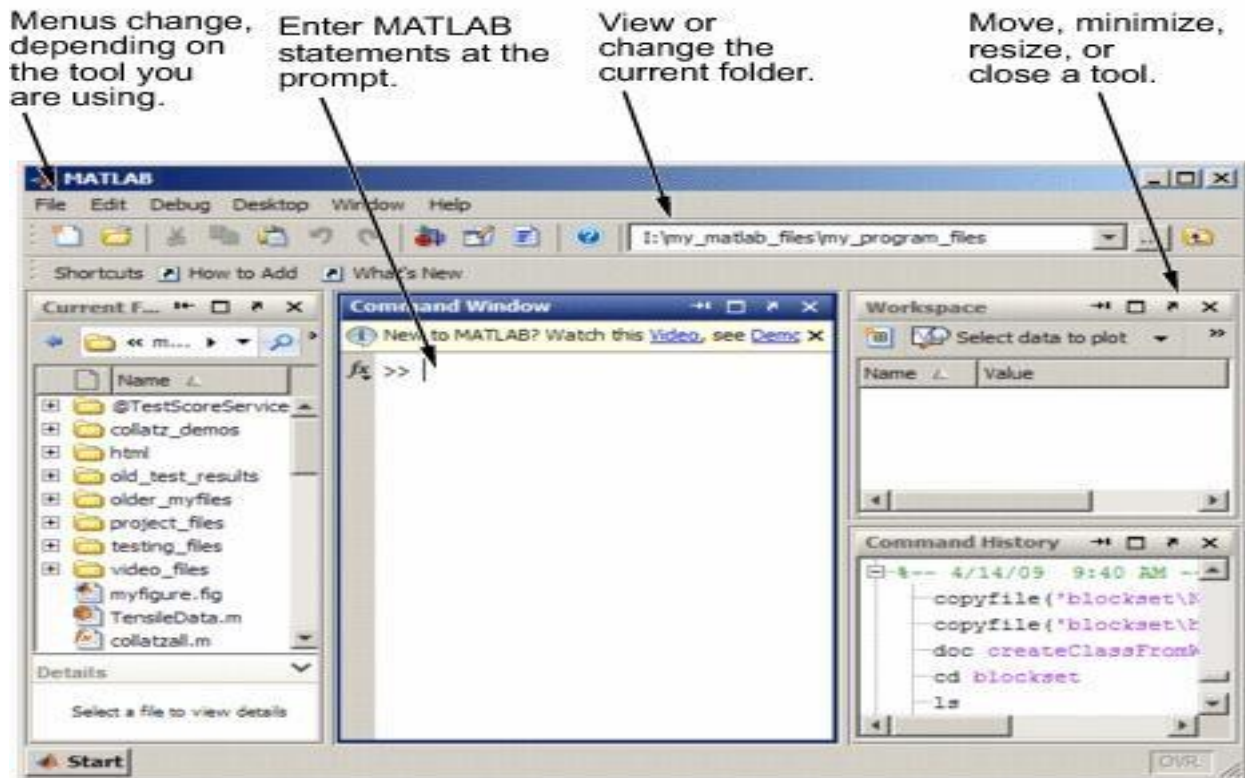
The external interfaces library allows you to write C/C++ and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), for calling MATLAB as a computational engine, and for reading and writing MAT-files.

## **Starting MATLAB**

After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut icon (MATLAB 7.0.4) on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows.

When you start MATLAB, the desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The following illustration shows the default desktop. You can customize the arrangement of tools and documents to suit your needs.



The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser
- The Start button

(`>>`) in the Command Window. Usually, there are 2 types of prompt:

- `>>` for full version
- `EDU>` for educational version

## Mathematical functions

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.

Typing `help elfun` and `help specfun` calls up full lists of elementary and special functions respectively.

There is a long list of mathematical functions that are built into MATLAB. These functions are called built-ins. Many standard mathematical functions, such as  $\sin(x)$ ,  $\cos(x)$ ,  $\tan(x)$ ,  $e^x$ ,  $\ln(x)$ , are evaluated by the functions `sin`, `cos`, `tan`, `exp`, and `log` respectively in MATLAB.

Here is the lists of some commonly used functions, where variables  $x$  and  $y$  can be numbers, vectors, or matrices.

### **Key Features**

The key features of MATLAB are:

- High-level language for technical computing
- Development environment for managing code, files, and data
- Interactive tools for iterative exploration, design, and problem solving
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration
- 2-D and 3-D graphics functions for visualizing data
- Tools for building custom graphical user interfaces
- Functions for integrating MATLAB based algorithms with external applications and languages, such as C, C++, Fortran, Java, COM, and Microsoft Excel

### **Elementary functions**

1.  $\cos(x)$  Cosine
2.  $\text{abs}(x)$  Absolute value
3.  $\sin(x)$  Sine
4.  $\text{sign}(x)$  Signum function
5.  $\tan(x)$  Tangent
6.  $\text{max}(x)$  Maximum value

7. `min(x)` Minimum value
8. `exp(x)` Exponential
9. `round(x)` Round to nearest integer
10. `sqrt(x)` Square

### **Getting help**

To view the online documentation, select MATLAB Help from Help menu or MATLAB Help directly in the Command Window. The preferred method is to use the Help Browser. The Help Browser can be started by selecting the ? Icon from the desktop toolbar. On the other hand, information about any command is available by typing

```
>> help Command
```

### **Quitting the MATLAB Program**

To end your MATLAB session, select File > Exit MATLAB in the desktop, or type quit in the Command Window. You can run a script file named finish.m each time MATLAB quits that, for example, executes functions to save the workspace.

### **Confirm Quitting**

MATLAB can display a confirmation dialog box before quitting. To set this option, select File > Preferences >

General > Confirmation Dialogs, and select the check box for Confirm before exiting MATLAB.



### **Advantages**

- MATLAB may behave as a calculator or as a programming language
- MATLAB combines nicely calculation and graphic plotting.
- MATLAB is relatively easy to learn

- MATLAB is interpreted (not compiled)
- MATLAB is optimized to be relatively fast when performing matrix operations
- MATLAB does have some object-oriented elements

### **Disadvantages**

- MATLAB is not a general purpose programming language such as C, C++, or FORTRAN
- MATLAB is designed for scientific computing, and is not well suitable for other applications
- MATLAB is an interpreted language, slower than a compiled language such as C++
- MATLAB commands are specific for MATLAB usage. Most of them do not have a direct equivalent with other programming language commands

### **POST EXPERIMENT QUESTIONS:**

1. What are five uses of MATLAB?
2. What data types does MATLAB use?
3. What are the five main parts of which the MATLAB system consists?

## LAB EXPERIMENT 2

### OBJECTIVE:

Write a program to perform the basics matrix operations.

### BRIEF DESCRIPTION:

Matrix operations play a crucial role in Matlab, as it is designed to work efficiently with arrays of data. Matlab provides a wide range of matrix operations that allow users to perform various computations and manipulations on matrices and vectors. Here are some key matrix operations in Matlab:

- 1. Matrix Creation:** Matlab provides different ways to create matrices. For example, you can create a matrix by specifying its elements explicitly, using the square brackets notation. You can also create special matrices like zeros, ones, or identity matrices using functions like `zeros()`, `ones()`, or `eye()`.
- 2. Matrix Arithmetic:** Matlab supports element-wise arithmetic operations on matrices, such as addition, subtraction, multiplication, and division. These operations are performed element by element, assuming the matrices have compatible dimensions. For matrix multiplication, you can use the `*` operator, or the matrix-specific function `mtimes()`.
- 3. Matrix Transpose:** The transpose operation swaps the rows and columns of a matrix. In Matlab, you can transpose a matrix using the `'` operator or the `transpose()` function.
- 4. Matrix Inversion:** Matlab provides the `inv()` function to compute the inverse of a square matrix. However, it is recommended to use more stable methods, such as the backslash operator `()` or the `pinv()` function, for solving linear systems or computing pseudo-inverses.
- 5. Matrix Concatenation:** Matlab allows you to concatenate matrices vertically or horizontally using square brackets `[]`. Vertical concatenation is performed using semicolons `(;)`, and horizontal concatenation is done using commas `(,)`.
- 6. Matrix Indexing and Slicing:** You can access specific elements or subsets of a matrix using indexing. Matlab uses parentheses `()` to index into matrices, and you can use indexing to retrieve individual elements, rows, columns, or submatrices.
- 7. Matrix Operations:** Matlab provides various built-in functions for matrix operations. For example, you can calculate the determinant using `det()`, compute the eigenvalues and eigenvectors using `eig()`, perform matrix exponentiation using `expm()`, calculate the matrix norm using `norm()`, and more.
- 8. Matrix Decompositions:** Matlab supports different matrix decompositions, such as LU decomposition, QR decomposition, and singular value decomposition (SVD). These decompositions are useful for solving linear systems, least-squares problems, and analyzing the properties of matrices.

---

These are just a few examples of the matrix operations available in Matlab. The software offers a rich set of functions and operators to manipulate and analyze matrices efficiently, making it a powerful tool for numerical computation and data analysis.

**PRE EXPERIMENT QUESTIONS:**

1. What are five uses of MATLAB?
2. What data types does MATLAB use?
3. What are the five main parts of which the MATLAB system consists?

**Explanation:**

```
>> magic(3) ans =
```

```
8     1     6
3     5     7
4     9     2
```

```
>> a=ones(3) a =
```

```
1     1     1
1     1     1
1     1     1
```

```
>> a=a*2 a =
```

```
2     2     2
2     2     2
2     2     2
```

```
>> b=2*a b =
```

```
4     4     4
4     4     4
4     4     4
```



```
>> operation=magic(4)
```

```
operation =
```

```
16    2    3    13
 5    11   10    8
 9    7    6    12
 4    14   15    1
```

```
>> operation(2,3)
```

```
ans = 10
```

```
>> operation(2:3,:)
```

```
ans =
```

```
5    11   10    8
 9    7    6    12
```

```
>> operation(:,2:3)
```

```
ans = 2    3
      11   10
      7    6
      14   15
```

```
>> operation(2:3,2:2)
```

```
ans = 11
```

```
7
```

**POST EXPERIMENT QUESTIONS:**

1. How to do matrix operations on MATLAB?
2. How do you precondition a matrix in MATLAB?
3. How do you solve a matrix equation in MATLAB?

---

## LAB EXPERIMENT 3

### OBJECTIVE:

How the weight & bias value effects the output of neurons.

### BRIEF DESCRIPTION:

In neural networks, the weight and bias values play a crucial role in determining the output of neurons. The weight values represent the strength or importance of the connections between neurons, while the bias values provide an additional parameter that helps in adjusting the output of neurons.

Here's how weight and bias values affect the output of neurons in a typical feed forward neural network:

**Weight Values:** The weight values determine the influence that the inputs to a neuron have on its output. Each input to a neuron is multiplied by its corresponding weight, and the resulting products are summed up. This sum represents the weighted input to the neuron. The weights essentially control the contribution of each input to the neuron's output. Larger weights amplify the input's effect, while smaller weights diminish it. By adjusting the weights, the neural network can learn to emphasize or de-emphasize certain inputs, allowing it to capture complex patterns and relationships in the data.

**Bias Values:** The bias values act as an offset or threshold for the neuron's activation. They represent the baseline activation level of the neuron, regardless of the inputs. The bias is added to the weighted sum of inputs, and the resulting value is passed through an activation function. The bias allows the neural network to shift the activation function along the input axis, affecting the overall output of the neuron. By adjusting the bias values, the network can control the overall output range and the threshold at which the neuron fires.

By adjusting the weight and bias values during the training process, neural networks can learn to approximate complex functions and make accurate predictions. The process of learning involves updating the weights and biases based on the error between the network's predicted output and the desired output. Through an optimization algorithm, such as backpropagation, the network iteratively adjusts the weights and biases to minimize the error and improve its performance on the given task.

In summary, weight and bias values in neural networks determine the strength of connections between neurons and the overall activation level of individual neurons. By adjusting these values, neural networks can learn to capture patterns and make predictions based on the provided inputs.

### PRE EXPERIMENT QUESTIONS:

1. What is the significance of weights and bias values?
2. What is the impact of weight and bias in neural network?

**Explanation:**

```
>> x=[1 2 3]
```

```
x =
```

```
1     2     3
```

```
>> w=[3 4 5]
```

```
w =
```

```
3     4     5
```

```
>> o=w'*x o =
```

```
3     6     9
```

```
4     8    12
```

```
5    10    15
```

```
>> b=1
```

```
b =
```

```
1
```

```
>> e=b*w' e =
```

```
3
```

```
4
```

```
5
```

```
>>
```

**POST EXPERIMENT QUESTIONS:**

1. How the weights and bias values affect the output of a neuron?
2. What is the importance of weights and bias values?

---

## LAB EXPERIMENT 4

### OBJECTIVE:

Write a program to plotting Activation Functions: Threshold functions, Signum function, Sigmoid function, Tan-hyperbolic function, Ramp function, Identity function using matlab

### BRIEF DESCRIPTION:

Activation functions play a crucial role in neural networks by introducing non-linearity to the model, enabling it to learn complex relationships between inputs and outputs. These functions are applied to the outputs of individual neurons or nodes in a neural network to determine their activation levels.

The sigmoid function maps the input to a range between 0 and 1, which is useful for binary classification problems. The ReLU function sets all negative input values to zero and keeps positive values unchanged. Leaky ReLU is a variation of the ReLU function that introduces a small slope for negative input values, avoiding dead neurons (neurons that never activate). The tanh function maps the input to a range between -1 and 1, providing stronger non-linearity than the sigmoid function. The softmax function is primarily used in the output layer of a neural network for multi-class classification problems. These are just a few examples of activation functions used in neural networks. Choosing the appropriate activation function depends on the nature of the problem, network architecture, and desired behavior of the model. Researchers continue to explore and develop new activation functions to improve the performance of neural networks.

### PRE EXPERIMENT QUESTIONS:

1. What do you understand by activation function?
2. What is the purpose of activation function?
3. What is an example of an activation function?

### Explanation:

```
% Activation Functions
```

```
x = linspace(-10, 10, 100);
```

```
% Threshold function
```

```
y_threshold = x >= 0;
```

```
% Signum function
```

```
y_signum = sign(x);
```

```
% Sigmoid function
```

```
y_sigmoid = 1 ./ (1 + exp(-x));
```

```
% Tanh function
```

```
y_tanh = tanh(x);
```

```
% Ramp function
```

```
y_ramp = max(0, x);
```

```
% Identity function
```

```
y_identity = x;
```

```
% Plotting
```

```
figure;
```

```
subplot(3, 2, 1);
```

```
plot(x, y_threshold, 'LineWidth', 2);
```

```
title('Threshold Function');
```

```
xlabel('x');
```

```
ylabel('f(x)');
```

```
subplot(3, 2, 2);
```

```
plot(x, y_signum, 'LineWidth', 2);
```

```
title('Signum Function');
```

```
xlabel('x');
```

```
ylabel('f(x)');
```

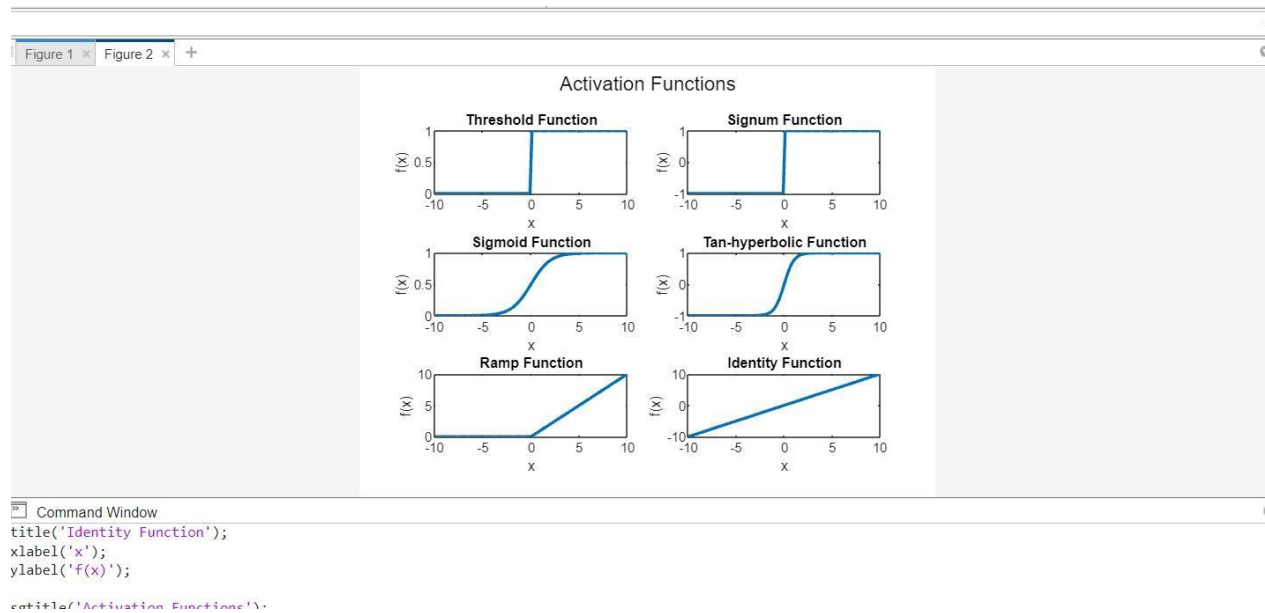
```
subplot(3, 2, 3);  
plot(x, y_sigmoid, 'LineWidth', 2);  
title('Sigmoid Function');  
xlabel('x');  
ylabel('f(x)');
```

```
subplot(3, 2, 4);  
plot(x, y_tanh, 'LineWidth', 2);  
title('Tan-hyperbolic Function');  
xlabel('x');  
ylabel('f(x)');
```

```
subplot(3, 2, 5);  
plot(x, y_ramp, 'LineWidth', 2);  
title('Ramp Function');  
xlabel('x');  
ylabel('f(x)');
```

```
subplot(3, 2, 6);  
plot(x, y_identity, 'LineWidth', 2);  
title('Identity Function');  
xlabel('x');  
ylabel('f(x)');
```

```
sgtitle('Activation Functions');
```

**Output:****POST EXPERIMENT QUESTIONS:**

1. How can we plot different activation function?
2. What are the advantages of activation functions?



---

## LAB EXPERIMENT 5

### OBJECTIVE:

Write a Program to implement MP Model.

### BRIEF DESCRIPTION:

The MP (McCulloch-Pitts) model, also known as the threshold logic unit, is a simplified model of a neuron in a neural network. It was introduced by Warren McCulloch and Walter Pitts in 1943 and served as a foundational concept for subsequent developments in artificial neural networks.

The MP model represents a binary neuron that takes multiple binary inputs and produces a binary output based on a fixed threshold. It captures the basic functionality of a real neuron by modeling the integration of input signals and the activation of the neuron based on a threshold.

Here are the key characteristics of the MP model:

1. **Binary Inputs:** The model takes binary inputs, where each input can be either 0 or 1. These inputs represent the input signals to the neuron.
2. **Weights:** Each input is associated with a weight, which determines the strength or importance of that input. The weights can be positive, negative, or zero.
3. **Threshold:** The model has a fixed threshold value, which is a predetermined threshold that the sum of the weighted inputs must exceed for the neuron to produce an output.
4. **Weighted Sum:** The inputs are multiplied by their respective weights, and the weighted sum of these inputs is computed.
5. **Activation Function:** The weighted sum is compared to the threshold using an activation function. If the weighted sum exceeds the threshold, the neuron fires and produces an output of 1; otherwise, it remains inactive and produces an output of 0.

The MP model provides a binary decision mechanism, making it suitable for tasks that require simple classification or decision-making. It forms the basis for more complex and powerful neural network models, such as the perceptron and artificial neural networks with multiple layers (multilayer perceptron), which allow for more flexible and sophisticated computations.

### PRE EXPERIMENT QUESTIONS:

1. What do you understand by MP neuron?
2. What do you understand by activation function?

**Explanation:**

```
% AND function using McCulloch-Pitts neuron
```

```
clear;
```

```
clc;
```

```
% Getting weights and threshold value
```

```
disp('Enter the weights');
```

```
w1=input('Weight w1=');
```

```
w2=input('Weight w2=');
```

```
disp('Enter threshold value');
```

```
theta=input('theta=');
```

```
y=[0 0 0 0];
```

```
x1=[1 1 0 0];
```

```
x2=[1 0 1 0];
```

```
z=[1 0 0 0];
```

```
con=1;
```

```
while con
```

```
    zin = x1*w1+x2*w2;
```

```
    for i=1:4
```

```
        if zin(i)>=theta
```

```
            y(i)=1;
```

```
        else y(i)=0;
```

```
        end
```

```
    end
```

```
    disp('Output of net=');
```

```
    disp(y);
```

```
    if y==z
```

```
        con=0;
```

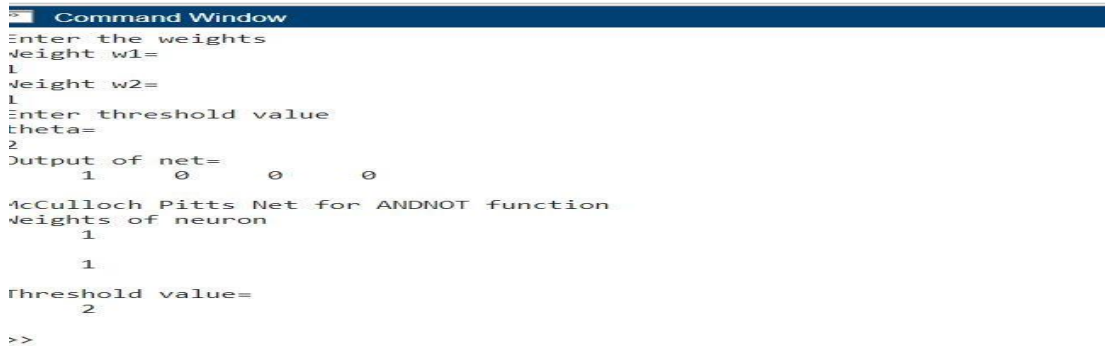
```
    else
```

```

    disp('Net is not learning Enter another set of weights and threshold value');
    w1=input('Weight w1=');
    w2=input('Weight w2=');
    theta=input('theta=');
    end
end
disp('McCulloch Pitts Net for ANDNOT function');
disp('Weights of neuron');
disp(w1);
disp(w2);
disp('Threshold value=');
disp(theta);

```

### **Output:**



```

Command Window
Enter the weights
weight w1=
1
weight w2=
1
Enter threshold value
theta=
2
Output of net=
1 0 0 0

McCulloch Pitts Net for ANDNOT function
Weights of neuron
1

Threshold value=
2
>>

```

### **POST EXPERIMENT QUESTIONS:**

1. What are the practical issues in neural network training?
2. What are the limitations of M-P neuron?
3. Why M-P neuron is widely used in processing binary data?

## LAB EXPERIMENT 6

### OBJECTIVE:

Write a program to generate ANDNOT function using McCulloch-Pitts neural net.

### BRIEF DESCRIPTION:

The McCulloch-Pitts neural network, proposed by Warren McCulloch and Walter Pitts in the 1940s, was one of the earliest models of an artificial neuron. Although it is a simplified representation of a biological neuron, it laid the foundation for modern artificial neural networks.

The McCulloch-Pitts neuron is a binary threshold logic unit that takes binary inputs and produces a binary output. The inputs can be either 0 or 1, and the output is determined by comparing the weighted sum of the inputs to a threshold value.

The ANDNOT function is a logical operation that takes two binary inputs, often represented as A and B, and produces an output based on the following rules:

- If A is 1 and B is 0, the output is 1.
- In all other cases ( $A = 0$  or  $B = 1$  or both), the output is 0.

To implement the ANDNOT function using a McCulloch-Pitts neural network, we can assign appropriate weights and thresholds to the inputs and combine them using the following steps:

1. Assign weights to the inputs: Let's assume we have two inputs, A and B. We assign a weight of 1 to input A and a weight of -1 to input B.
2. Set the threshold: In this case, we set the threshold to 0.
3. Compute the weighted sum: Multiply each input by its corresponding weight and sum the results. Let's denote the weighted sum as S. For the ANDNOT function, the weighted sum S is calculated as  $S = A1 + B(-1)$ .
4. Apply the threshold logic: Compare the weighted sum S to the threshold. If S is greater than or equal to the threshold, the output is 1; otherwise, the output is 0. In this case, if  $S \geq 0$ , the output is 1, and if  $S < 0$ , the output is 0.
5. By following these steps, we have constructed a McCulloch-Pitts neural network that performs the ANDNOT function. This network can take binary inputs A and B, apply weights and a threshold, and produce an output that follows the logic defined by the ANDNOT function.

### PRE EXPERIMENT QUESTIONS:

1. What is M-P Neuron model?
2. What do you understand by activation function?
3. What is the truth table of ANDNOT function?

**Explanation:**

```
% ANDNOT function using McCulloch-Pitts neuron
```

```
clear;
```

```
clc;
```

```
% Getting weights and threshold value
```

```
disp('Enter the weights');
```

```
w1=input('Weight w1=');
```

```
w2=input('Weight w2=');
```

```
disp('Enter threshold value');
```

```
theta=input('theta=');
```

```
y=[0 0 0 0];
```

```
x1=[0 0 1 1];
```

```
x2=[0 1 0 1];
```

```
z=[0 0 1 0];
```

```
con=1;
```

```
while con
```

```
zin = x1*w1+x2*w2;
```

```
for i=1:4
```

```
if zin(i)>=theta
```

```
y(i)=1;
```

```
else y(i)=0;
```

```
end
```

```
end
```

```
disp('Output of net=');
```

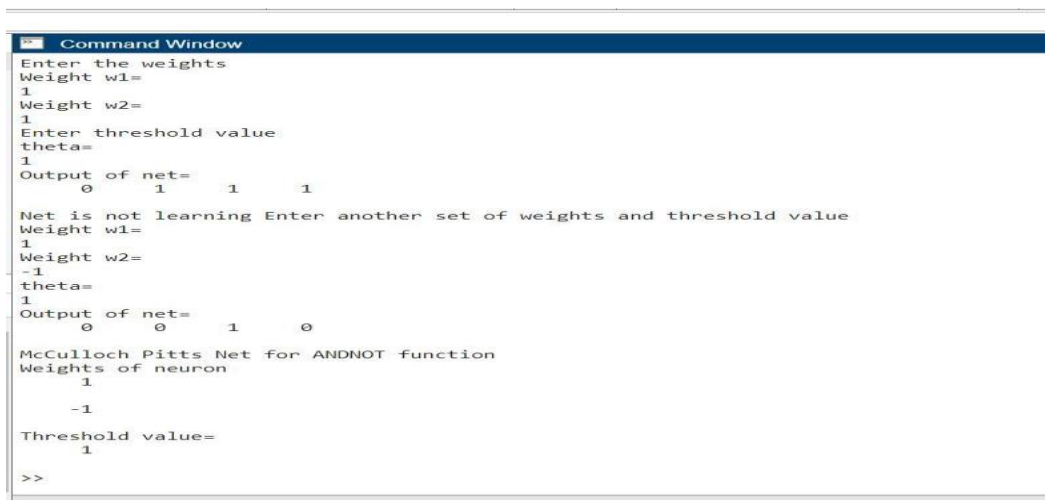
```
disp(y);
```

```
if y==z
```

```
con=0;
```

```
else
disp('Net is not learning Enter another set of weights and threshold value');
w1=input('Weight w1=');
w2=input('Weight w2=');
thete=input('theta=');
end
end
disp('McCulloch Pitts Net for ANDNOT function');
disp('Weights of neuron');
disp(w1);
disp(w2);
disp('Threshold value=');
disp(theta);
```

### **Output:**



```
Command Window
Enter the weights
Weight w1=
1
Weight w2=
1
Enter threshold value
theta=
1
Output of net=
0 1 1 1

Net is not learning Enter another set of weights and threshold value
Weight w1=
1
Weight w2=
-1
theta=
1
Output of net=
0 0 1 0

McCulloch Pitts Net for ANDNOT function
Weights of neuron
1
-1
Threshold value=
1
>>
```

### **POST EXPERIMENT QUESTIONS:**

1. How to implement ANDNOT using MP Neuron Model.
2. What is significant of implementing MP Neuron Model with ANDNOT.

## LAB EXPERIMENT 7

### OBJECTIVE:

Write a program to generate XOR function using McCulloch-Pitts neural net

### BRIEF DESCRIPTION:

The XOR (exclusive OR) function is a logical operation that takes two binary inputs, often denoted as A and B, and produces an output based on the following rules:

- If A and B are both 0 and both 1, the output is 0.
- If A is 0 and B is 1, or A is 1 and B is 0, the output is 1.

Implementing the XOR function using a single-layer McCulloch-Pitts neural network is not possible because the XOR function is not linearly separable. However, we can combine multiple McCulloch-Pitts neurons to create a multi-layer network that can accurately represent the XOR function.

Let's consider a two-layer network with three neurons: two input neurons (A and B) and one output neuron (Y). We can set up the network as follows:

1. Assign weights and thresholds to the neurons:

- For the input neurons A and B, assign a weight of 1 to both.
- For the output neuron Y, assign a weight of -1 to both A and B.

2. Compute the weighted sum at each neuron:

- At neuron A, compute the weighted sum by multiplying the input A by its weight (1).
- At neuron B, compute the weighted sum by multiplying the input B by its weight (1).
- At neuron Y, compute the weighted sum by multiplying A and B by their weights (-1) and summing the results. The weighted sum at neuron Y is given by  $S = A*(-1) + B*(-1)$ .

3. Apply the threshold logic:

- For the input neurons A and B, the output is simply their binary values (0 or 1).
- For the output neuron Y, compare the weighted sum S to the threshold. If S is greater than or equal to the threshold (0 in this case), the output is 1; otherwise, the output is 0.

By following these steps, we have constructed a two-layer McCulloch-Pitts neural network that can perform the XOR function. This network can take binary inputs A and B, apply weights and thresholds, and produce an output that follows the logic defined by the XOR function. The introduction of the second layer allows for non-linear decision boundaries and enables the network to accurately represent the XOR function.

### PRE EXPERIMENT QUESTIONS:

1. What is M-P Neuron model?

2. What do you understand by activation function?
3. What is the truth table of XOR function?

**Explanation:**

```
% XOR function using McCulloch-Pitts neuron
```

```
clear;
```

```
clc;
```

```
% Getting weights and threshold value
```

```
disp('Enter the weights');
```

```
w11=input('Weight w11=');
```

```
w12=input('Weight w12=');
```

```
w21=input('Weight w21=');
```

```
w22=input('Weight w22=');
```

```
v1=input('Weight v1=');
```

```
v2=input('Weight v2=');
```

```
disp('Enter threshold value');
```

```
theta=input('theta=');
```

```
x1=[0 0 1 1];
```

```
x2=[0 1 0 1];
```

```
z=[0 1 1 0];
```

```
con=1;
```

```
while con
```

```
zin1 = x1*w11+x2*w21;
```

```
zin2 = x1*w21+x2*w22;
```

```
for i=1:4
```

```
if zin1(i)>=theta
```

```
y1(i)=1;
```

```
else y1(i)=0;
```



---

```
end
if zin2(i)>=theta
y2(i)=1;
else y2(i)=0;
end
end
yin=y1*v1+y2*v2;
for i=1:4
if yin(i)>=theta;
y(i)=1;
else
y(i)=0;
end
end
disp('Output of net=');
disp(y);
if y==z
con=0;
else
disp('Net is not learning Enter another set of weights and threshold value');
w11=input('Weight w11=');
w12=input('Weight w12=');
w21=input('Weight w21=');
w22=input('Weight w22=');
v1=input('Weight v1=');
v2=input('Weight v2=');
theta=input('theta=');
end
```

```
end
disp('McCulloch Pitts Net for XOR function');
disp('Weights of neuron Z1');
disp(w11);
disp(w21);
disp('Weights of neuron Z2');
disp(w12);
disp(w22);
disp('Weights of neuron Y');
disp(v1);
disp(v2);
disp('Threshold value=');
disp(theta);
```

### **Output:**



```
Command Window
Weight w22=
1
Weight v1=
1
Weight v2=
1
Enter threshold value
theta=
1
Output of net=
0 1 1 0

McCulloch Pitts Net for XOR function
Weights of neuron Z1
1
-1

Weights of neuron Z2
-1
1

Weights of neuron Y
1
1

Threshold value=
1

>>
```

### **POST EXPERIMENT QUESTIONS:**

1. How to implement XOR using MP Neuron Model.
2. What is the significance of implementing MP Neuron Model with XOR.

## LAB EXPERIMENT 8

### OBJECTIVE:

Write a Program to Implement Hebb Model.

### BRIEF DESCRIPTION:

The Hebbian learning rule, proposed by Donald Hebb in 1949, is a simple and fundamental principle in neural network learning. It states that "cells that fire together wire together," meaning that the strength of the connection between two neurons increases when they are activated simultaneously.

The Hebbian learning rule is based on the idea of synaptic plasticity, which refers to the ability of the connections (synapses) between neurons to change in strength. It provides a mechanism for learning and memory formation in neural networks.

The basic principle of the Hebbian learning rule can be summarized as follows:

1. If two connected neurons are both activated (fire) at the same time, the strength of the connection between them is strengthened.
2. If two connected neurons are not activated together, or only one of them is activated, the connection between them remains unchanged or weakens.

This rule is often expressed mathematically as:

$$\Delta W = \eta * X * Y$$

- $\Delta W$  represents the change in synaptic weight (connection strength).
- $\eta$  (eta) is the learning rate or the scaling factor that determines the magnitude of weight change.
- $X$  and  $Y$  are the activities (outputs) of the pre-synaptic and post-synaptic neurons, respectively.

The Hebbian learning rule can be applied in a single neuron or in a network of neurons. It allows the network to learn associations between input patterns and modify its connection strengths accordingly.

It's important to note that while the Hebbian learning rule is a fundamental concept in neural network learning, it has certain limitations and can lead to instability or over fitting in certain scenarios. Therefore, modern neural network architectures often employ more advanced learning algorithms, such as back propagation, that address these limitations and offer more efficient and robust learning mechanisms.

**PRE EXPERIMENT QUESTIONS:**

1. What is the Hebb rule also known as?
2. What is Hebb learning more suited for?

**Explanation:**

```
% Hebbian Learning Rule Implementation
% Define the input patterns (binary values)
input_patterns = [0 0; 0 1; 1 0; 1 1];
% Define the target output pattern
target_output = [0; 1; 1; 0];
% Initialize the synaptic weights with random values
synaptic_weights = rand(1, 2);
% Set the learning rate
learning_rate = 0.1;
% Set the number of training epochs
epochs = 100;
% Perform training
for epoch = 1:epochs
    % Iterate through each input pattern
    for p = 1:size(input_patterns, 1)
        % Retrieve the current input pattern and target output
        input_pattern = input_patterns(p, :);
        target = target_output(p);
        % Compute the net input (weighted sum)
        net_input = input_pattern * synaptic_weights';
        % Apply the activation function (Heaviside step function)
        output = double(net_input >= 0);
        % Update the synaptic weights using the Hebbian learning rule
        synaptic_weights = synaptic_weights + learning_rate * input_pattern * (target - output);
    end
end
```

```
end
end

% Display the learned synaptic weights
disp('Learned Synaptic Weights:');
disp(synaptic_weights);

% Test the network
disp('Testing the Network:');
for p = 1:size(input_patterns, 1)
    % Retrieve the current input pattern and target output
    input_pattern = input_patterns(p, :);
    target = target_output(p);

    % Compute the net input (weighted sum)
    net_input = input_pattern * synaptic_weights';

    % Apply the activation function (Heaviside step function)
    output = double(net_input >= 0);

    % Display the input pattern, target output, and network output
    disp(['Input: ' num2str(input_pattern) ' Target: ' num2str(target) ' Output: ' num2str(output)]);
end
```

### **Output:**

```
% Display the input pattern, target output, and network output
disp(['Input: ' num2str(input_pattern) ' Target: ' num2str(target) ' Output: ' num2str(output)]);
end
Learned Synaptic Weights:
-0.0853 -0.0942

Testing the Network:
Input: 0 0 Target: 0 Output: 1
Input: 0 1 Target: 1 Output: 0
Input: 1 0 Target: 1 Output: 0
Input: 1 1 Target: 0 Output: 0
>>
```

**POST EXPERIMENT QUESTIONS:**

1. Is Hebbian learning inspired by the weight adjustment mechanism?
2. Is Hebbian learning supervised or unsupervised?

---

## LAB EXPERIMENT 9

### OBJECTIVE:

How the choice of activation function affect the output of neuron. Experiment with the following

- (i) Purelin
- (ii) Binary Thresold (Hardlim, Hardlims)
- (iii) Sigmoid (logsig,tansig)

### BRIEF DESCRIPTION:

The choice of activation function can have a significant impact on the output of a neuron in a neural network. The activation function determines the output of a neuron based on its weighted sum of inputs. It introduces non-linearity into the network and allows the model to learn complex patterns and relationships in the data.

Here are some common activation functions and their effects on the output of a neuron:

**Sigmoid Function:** The sigmoid function maps the input to a value between 0 and 1. It has a smooth, S-shaped curve. When the input is small or large, the output tends to saturate, resulting in gradients close to zero. This saturation can make training slower, especially in deep networks. However, the sigmoid function is useful when you need to model a probability-like output.

**Hyperbolic Tangent (tanh) Function:** The tanh function is similar to the sigmoid function but maps the input to a value between -1 and 1. Like the sigmoid function, it suffers from saturation at extreme values. However, it is symmetric around the origin and is often preferred over the sigmoid function when the output range needs to be centered around zero.

**Rectified Linear Unit (ReLU) Function:** The ReLU function sets the output to zero for negative inputs and keeps the output linear for positive inputs. It is the most widely used activation function due to its simplicity and computational efficiency. ReLU helps in mitigating the vanishing gradient problem and has been found to be effective in deep neural networks. However, ReLU can also lead to dead neurons if their weights are not updated properly, resulting in neurons that never activate.

**Leaky ReLU:** The leaky ReLU is an extension of the ReLU function that allows a small slope for negative inputs. Instead of being zero, the output is a small fraction (e.g., 0.01) of the input. This modification helps alleviate the dying ReLU problem by allowing a small gradient for negative inputs, which enables the neuron to recover from negative saturation.

**Softmax Function:** The softmax function is commonly used in the output layer of a neural network for multi-class classification tasks. It normalizes the outputs such that they represent probabilities, ensuring that the sum of all output values is equal to 1. This function is useful when you want to determine the class probabilities of mutually exclusive classes.

The choice of activation function should be based on the specific problem at hand and the characteristics of the data. Experimentation and empirical evaluation are often necessary to determine the most suitable activation function for a particular task.

**PRE EXPERIMENT QUESTIONS:**

1. What is the principle of activation function?
2. Why is it called activation function?
3. Why is activation function nonlinear?

**Explanation:**

```
>> x=[1,2,3]
```

```
x =
```

```
1     2     3
```

```
>> w=[4,5,6]
```

```
w =
```

```
4     5     6
```

```
>> o=x*w'
```

```
o =
```

```
32
```

```
>> b=[1]
```

```
b =
```

```
1
```

```
>> e=b*w'
```

```
e =
```

```
4
```

```
5
```

```
6
```

```
>> n=o+e
```

```
n =
```



36

37

38

```
>> hardlim(n)
```

```
ans =
```

```
1
```

```
1
```

```
1
```

```
>> hardlims(n)
```

```
ans =
```

```
1
```

```
1
```

```
1
```

```
>> purelin(n)
```

```
ans =
```

```
36
```

```
37
```

```
38
```

```
>> logsig(n)
```

```
ans =
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
>> tansig(n)
```

```
ans =
```

```
1
```

```
1
```

```
1
```

**POST EXPERIMENT QUESTIONS:**

1. Why use ReLU instead of linear?
2. What is the best activation function in neural networks?
3. How the choices of activation function affect the output of neuron?

---

## LAB EXPERIMENT 10

### OBJECTIVE:

How the Perceptron Learning rule works for Linearly Separable Problem.

### BRIEF DESCRIPTION:

The Perceptron Learning rule is a basic algorithm used for training a type of artificial neural network called a perceptron. It is designed to solve linearly separable problems, which are classification problems where a linear decision boundary can accurately separate the classes.

Here's a brief description of how the Perceptron Learning rule works for a linearly separable problem:

1. Initialization: Initially, the weights and bias of the perceptron are set to small random values or zero.
2. Input and Activation: Each input feature is multiplied by its corresponding weight, and all the weighted inputs are summed up with the bias term. This sum is then passed through an activation function (commonly a step function) to produce the output of the perceptron.
3. Error Calculation: The output of the perceptron is compared to the desired output (target class label). If the output matches the target, there is no error. Otherwise, there is a prediction error, and the weights need to be adjusted.
4. Weight Update: The weights and bias of the perceptron are updated based on the prediction error. The weight update rule is as follows:
  - If the output is too low (0) while the target is 1, the weights are increased by adding the input to the current weight values.
  - If the output is too high (1) while the target is 0, the weights are decreased by subtracting the input from the current weight values.
5. Iteration: Steps 2-4 are repeated for each training example until the perceptron achieves a satisfactory level of accuracy or until a maximum number of iterations is reached.
6. Convergence: The perceptron learning rule guarantees convergence to a solution if the problem is linearly separable. This means that if there exists a linear decision boundary that can perfectly separate the classes, the perceptron will eventually find it through weight updates.
7. Output: Once the perceptron has converged, it can be used to classify new unseen examples by applying the learned weights and bias to the input features and passing them through the activation function.

It's important to note that the Perceptron Learning rule is limited to solving linearly separable problems and cannot handle problems that require non-linear decision boundaries. For such

problems, more advanced neural network architectures like multi-layer perceptrons (MLPs) or deep learning models are used.

### PRE EXPERIMENT QUESTIONS:

1. What do you understand by linear separable problem?
2. What is an example of linear separability perceptron?

### Explanation:

```
% Generate random linearly separable data
numSamples = 100; % Number of data points
dim = 2; % Data dimension
classA = [randn(dim, numSamples/2) + repmat([2; 2], 1, numSamples/2)];
classB = [randn(dim, numSamples/2) + repmat([-2; -2], 1, numSamples/2)];
data = [classA, classB];
labels = [ones(1, numSamples/2), -ones(1, numSamples/2)];

% Initialize weights and bias
weights = rand(dim, 1);
bias = rand();

% Set learning rate and maximum number of iterations
learningRate = 0.1;
maxIterations = 1000;

% Perceptron Learning Rule
iteration = 0;
misclassified = true;
while misclassified && iteration < maxIterations
    misclassified = false;
```

---

```
% Iterate through each data point
for i = 1:numSamples
    input = data(:, i);
    target = labels(i);

    % Compute the output of the perceptron
    output = sign(weights' * input + bias);

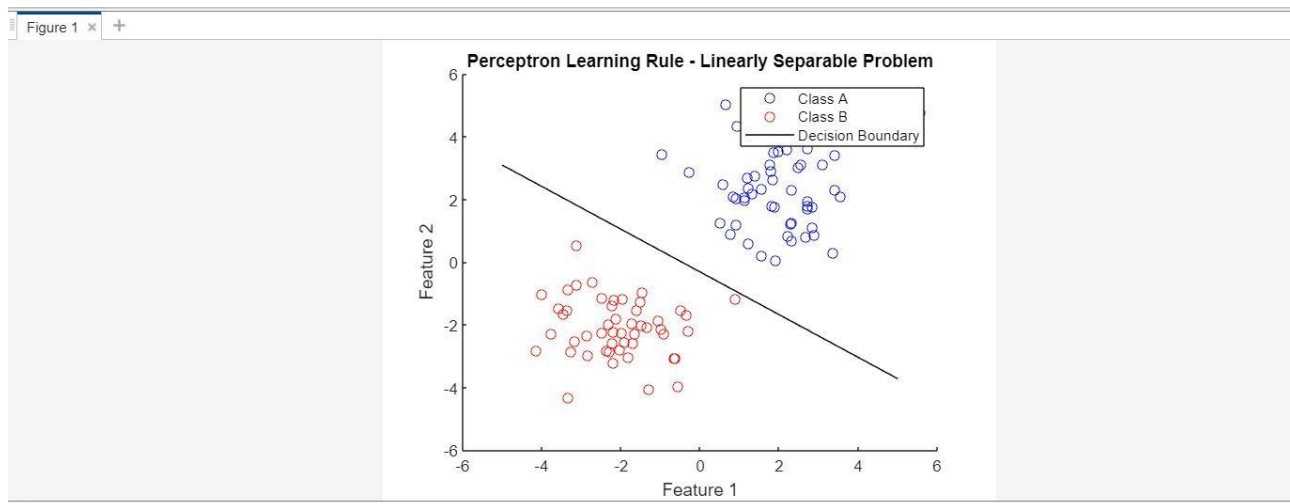
    % Update weights and bias if misclassified
    if output ~= target
        weights = weights + learningRate * target * input;
        bias = bias + learningRate * target;
        misclassified = true;
    end
end

iteration = iteration + 1;
end

% Plot the data points and decision boundary
figure;
hold on;
scatter(classA(1, :), classA(2, :), 'b');
scatter(classB(1, :), classB(2, :), 'r');
x = linspace(-5, 5);
y = (-weights(1) * x - bias) / weights(2);
plot(x, y, 'k');
xlabel('Feature 1');
```

```
ylabel('Feature 2');  
title('Perceptron Learning Rule - Linearly Separable Problem');  
legend('Class A', 'Class B', 'Decision Boundary');  
hold off;
```

## OUTPUT



## POST EXPERIMENT QUESTIONS:

1. How the perceptron algorithm works in solving the linear problem?
2. Does perceptron only work for linearly separable data?

## LAB EXPERIMENT 11

### OBJECTIVE:

. How the Perceptron Learning rule works for Non-Linearly Separable Problem.

### BRIEF DESCRIPTION:

The Perceptron learning rule is a simple algorithm used for training a binary classifier, specifically a single-layer neural network called a perceptron. The algorithm works by adjusting the weights of the perceptron based on the training data until it can accurately classify the given inputs.

However, the Perceptron learning rule has a limitation—it can only learn to classify linearly separable problems. Linearly separable problems are those in which it is possible to draw a straight line (in two dimensions) or a hyper plane (in higher dimensions) to separate the two classes.

When faced with non-linearly separable problems, the Perceptron learning rule may not converge and find a solution. In these cases, the perceptron will keep updating the weights indefinitely without reaching a satisfactory solution.

To address this limitation, various techniques can be employed to handle non-linearly separable problems. When faced with non-linearly separable problems, the Perceptron learning rule is not sufficient on its own. Techniques like feature engineering, multilayer perceptron, support vector machines, or other non-linear classifiers should be considered to handle such problems effectively.

### PRE EXPERIMENT QUESTIONS:

1. What is Perceptron Learning rule?
2. What do you understand by non-linearly separable problem?
3. Can perceptron solve non linearly separable problems?

### Explanation:

```
% Perceptron Learning Rule for Non-Linearly Separable Problem
```

```
% Define the input patterns
```

```
X = [-1 -1; -1 1; 1 -1; 1 1];
```

```
% Define the corresponding target outputs
```

```
T = [0; 1; 1; 0];
```

```
% Initialize the weights and biases
```

```
w1 = randn();
```

```
w2 = randn();
```

---

```
b = randn();

% Set the learning rate and maximum number of epochs
learningRate = 0.1;
maxEpochs = 1000;

% Define the sigmoid activation function
sigmoid = @(x) 1./(1 + exp(-x));

% Train the Perceptron
for epoch = 1:maxEpochs
    errorCount = 0;
    for i = 1:size(X, 1)
        % Forward pass
        input = X(i, :);
        net = w1 * input(1) + w2 * input(2) + b;
        output = sigmoid(net);

        % Compute the error
        error = T(i) - output;
        if error ~= 0
            errorCount = errorCount + 1;
        end

        % Update the weights and bias
        w1 = w1 + learningRate * error * input(1);
        w2 = w2 + learningRate * error * input(2);
        b = b + learningRate * error;
    end
end
```



```
end

% Check for convergence
if errorCount == 0
    fprintf('Converged at epoch %d\n', epoch);
    break;
end
end

% Display the final weights and bias
fprintf('Final weights: w1=%.2f, w2=%.2f, bias=%.2f\n', w1, w2, b);
```

**OUTPUT:**

```
>> untitled
Final weights: w1=-0.03, w2=-0.05, bias=0.00
>>
```

**POST EXPERIMENT QUESTIONS:**

- 1 Does the perceptron algorithms converge is the examples are not linearly separable?
2. How the Perceptron Learning rule works for Non-Linearly Separable Problem.

This lab manual has been updated by

Prof Renu Narwal

([renu@ggnindia.dronacharya.info](mailto:renu@ggnindia.dronacharya.info))

Crosschecked By HOD CSE

Please spare some time to provide your valuable feedback.