318556: Software Lab-II (Artificial Neural Network) Laboratory

Second Year (2020 Course)

Semester - II

Teaching Scheme	Examination Scheme
Pract	25 Marks
	50 Marks

LABORATORY MANUAL

DEPARTMENT OF AIML Engineering

Samarth College of Engineering & Management, Pune

2024-2025

.

INDEX

Sr. Sr. No.	Title of Experiment	Page No.	Marks	Sign
1. 1.	**Group B** Write a program to scheme a few activation functions that are used in neural networks			
2.	Write a program to show back propagation network for XOR function with binary input and output			
3.	Write a program for producing back propagation feed forward network			
4.	Write a program to demonstrate ART			
5.	Write a program to demonstrate the perceptron learning law with its decision region using python. Give the output in graphical form			

*Certified that Mr/Miss*____ *of*

Class ________Sem_______Roll no.______has completed the term work satisfactorily in the subject____________________________________of the Department_______________of SGOI College of Engineering & Management Belhe, During academic year.

Staff Member   Head of Dept.

Prof. Shelake S. D.

GROUP - B

EXPERIMENT NO. 1

- **Aim:** Write a program to scheme a few activation functions that are used in neural networks.
- **Outcome:** At end of this experiment, student will be able to Write a Python program to plot a few activation functions that are being used in neural networks.
- **Hardware Requirement:**
- **Software Requirement:** Ubuntu OS, Python Editor(Python Interpreter)
- **Theory:**
  In the process of building a neural network, one of the choices you get to make is what Activation Function to use in the hidden layer as well as at the output layer of the network.

Elements of a Neural Network

Input Layer: This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

Hidden Layer: Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

Output Layer: This layer bring up the information learned by the network to the outer world.

What is an activation function and why use them?

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

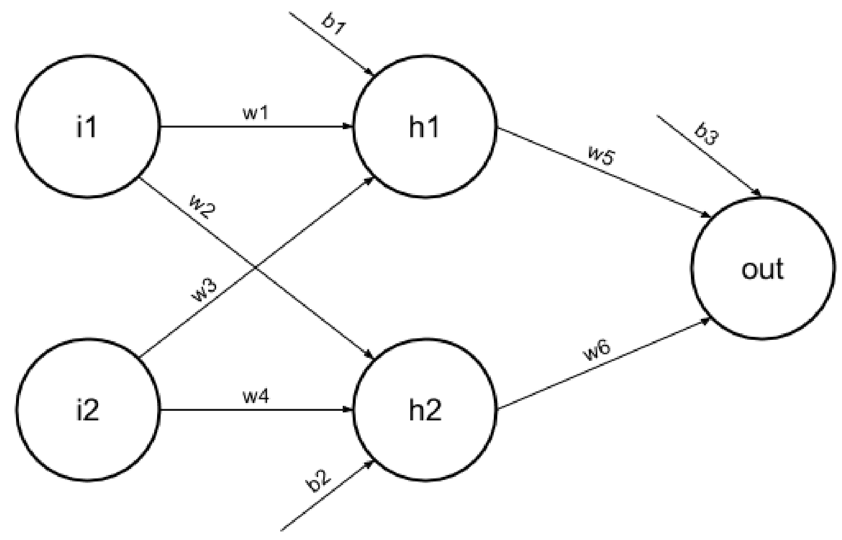
Explanation: We know, the neural network has neurons that work in correspondence with weight, bias, and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the backpropagation possible since the gradients are supplied along with the error to update the weights and biases.

Why do we need Non-linear activation function?

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Mathematical proof

Suppose we have a Neural net like this :-



Elements of the diagram are as follows:

Hidden layer i.e. layer 1:

z(1) = W(1)X + b(1) a(1)

Here,

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

(Note: We are not considering activation function here) Layer 2

i.e. output layer :-

Note : Input for layer 2 is output from layer 1 z(2)

= W(2)a(1) + b(2)

a(2) = z(2)

Calculation at Output layer

z(2) = (W(2) * [W(1)X + b(1)]) + b(2)

z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]

Let,

[W(2) * W(1)] = W

[W(2)*b(1) + b(2)] = b

Final output : z(2) = W*X + b which is again a linear function

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because *the composition of two linear function is a linear function itself*. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. Hence we need an activation function.

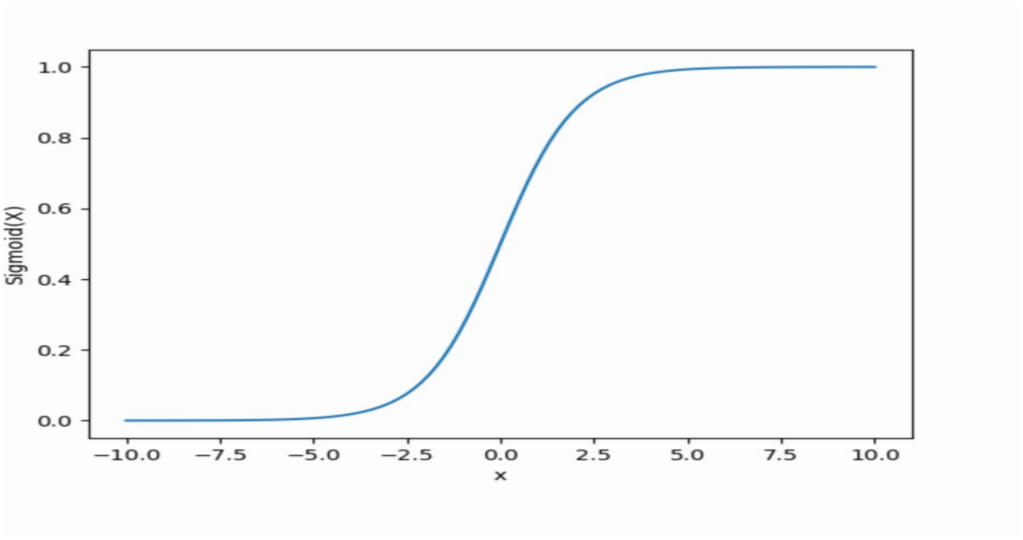
Variants of Activation Function

Linear Function

- Equation : Linear function has the equation similar to as of a straight line i.e. y = x
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- Range : -inf to +inf
- Uses : Linear activation function is used at just one place i.e. output layer.
- Issues : If we will differentiate linear function to bring non-linearity, result will no more depend on *input "x"* and function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

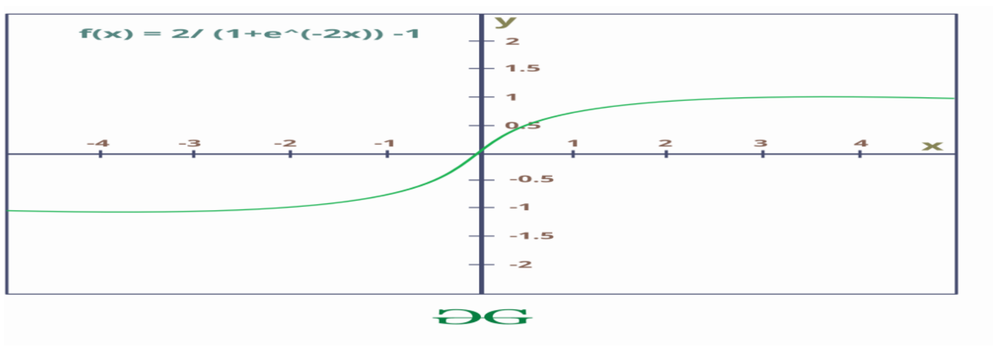
For example : Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

Sigmoid Function



- It is a function which is plotted as 'S' shaped graph.
- Equation : A = 1/(1 + e-x)
- Nature : Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- Value Range : 0 to 1
- Uses : Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be *1* if value is greater than 0.5 and *0* otherwise.

Tanh Function



- The activation that works almost always better than sigmoid function is Tanh function also known as Tangent Hyperbolic function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

- Equation :-

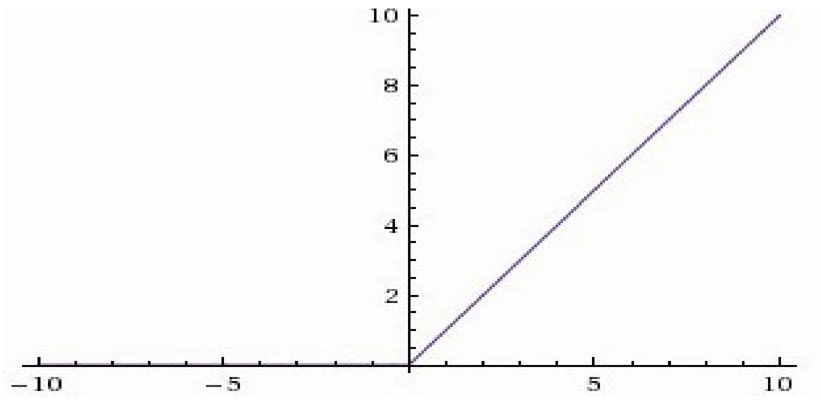
$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- Value Range :- -1 to +1

- Nature :- non-linear

- Uses: - Usually used in hidden layers of a neural network as its values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

RELU Function



- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.

- Equation :- *A(x) = max(0,x)*. It gives an output x if x is positive and 0 otherwise.

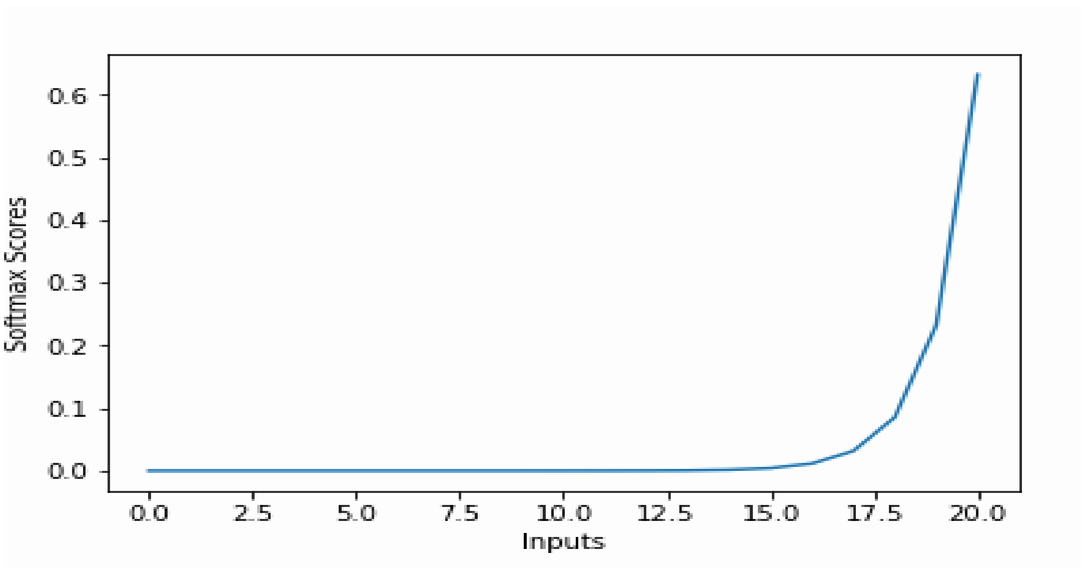
- Value Range :- [0, inf)

- Nature :- non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.

- Uses :- ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

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

Softmax Function



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

- Nature :- non-linear

- Uses :- Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

- Output:- The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

- The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.

- If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.

- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Conclusion: -

____

____

____

____

____

Questions:

Q1. What is the role of the Activation functions in Neural Networks?

Q2. List down the names of some popular Activation Functions used in Neural Networks?

Q3. How to initialize Weights and Biases in Neural Networks? Q4.

How are *Neural Networks* modelled?

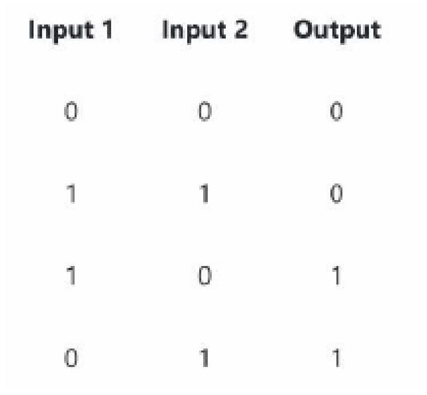
Q5. What is an *Activation Function*?

EXPERIMENT NO. 2

- **Aim:** Write a program to show back propagation network for XOR function with binary input and output
- **Theory:**

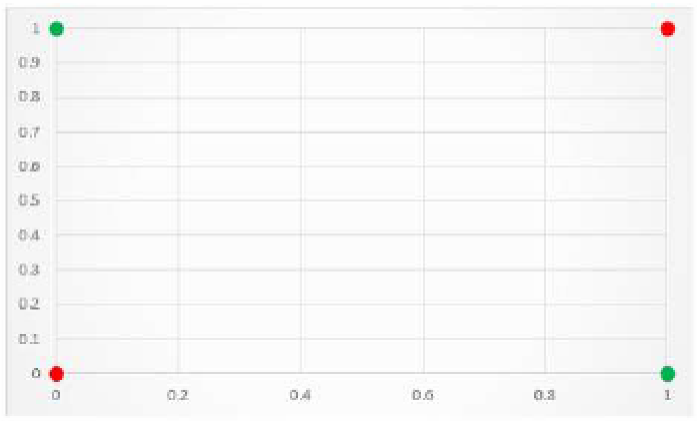
Solving XOR problem with Back Propagation Algorithm

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

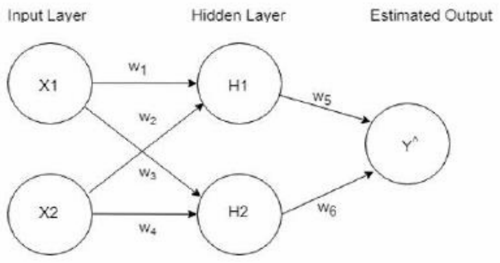
On the surface, XOR appears to be a very simple problem, however, Minksy and Papert (1969) showed that this was a big problem for neural network architectures of the 1960s, known as perceptrons. A limitation of this architecture is that it is only capable of separating data points with a single line.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



Solution

The back propagation 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.

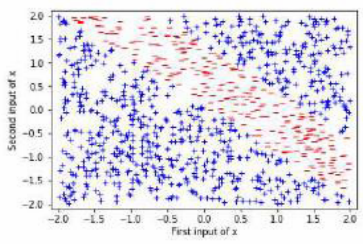
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Loss function:

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

Results

Classification WITHOUT gaussian noise



```
(731, 269)
```

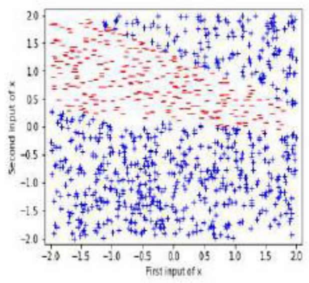
Observation : To classify, I generated 1000 x 2 random floats in range -2 to 2. Using weights from trained model, I classified each input & plotted it on 2-D space. Out of 1000, 731 points were classified as"+1" and 269 points were classified as **"-1"** It is clearly seen, classification region is not 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.

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

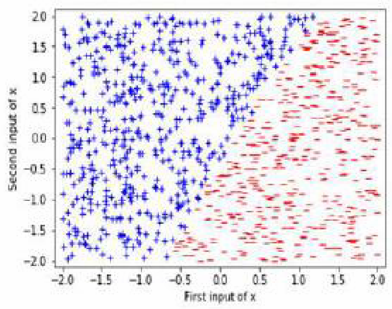
σ = 0.5



```
(702, 298)
```

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

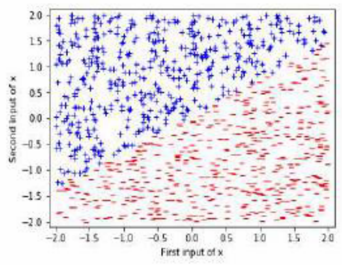
σ = 1.0



```
(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 σ = 1). Here also, we observe two distinct regions of classification.

Conclusion:

We have successfully implemented XOR problem with Back Propagation Algorithm.

Questions:

1. Explain two basic features of mapping.
2. Explain Feature Mapping Models.
3. Explain Self Organization Map.
4. Explain SOM Algorithm.
5. Explain properties of Feature Map.

EXPERIMENT NO. 3

- **Aim:** Write a program for producing back propagation feed forward network
- **Outcome:** At end of this experiment, student will be able to creating a Back Propagation Feed-forward neural network.
- **Hardware Requirement:**
- **Software Requirement:** Python IDE
- **Theory:**
- **Backpropagation Algorithm**

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks.Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell's axon to other cell's dendrites.The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state. Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer.Backpropagation can be used for both classification and regression problems, but we will focus on classification in this tutorial.

In classification problems, best results are achieved when the network has one neuron in the output layer for each class value. For example, a 2-class or binary classification problem with the class values of A and B. These expected outputs would have to be transformed into binary vectors with one column for each class value. Such as [1, 0] and [0, 1] for A and B respectively. This is called a one hot encoding. This tutorial is broken down into 5 parts:

1. Initialize Network.
2. Forward Propagate.
3. Back Propagate Error.
4. Train Network.
5. Predict.

1. Initialize Network

Let's start with something easy, the creation of a new network ready for training.
Each neuron has a set of weights that need to be maintained. One weight for each input connection and an additional weight for the bias. We will need to store additional properties for a neuron during training, therefore we will use a dictionary to represent each neuron and store properties by names such as '**weights**'for the weights.A network is organized into layers. The input layer is

really just a row from our training dataset. The first real layer is the hidden layer. This is followed by the output layer that has one neuron for each class value.
We will organize layers as arrays of dictionaries and treat the whole network as an array of layers. It is good practice to initialize the network weights to small random numbers. In this case, will we use random numbers in the range of 0 to 1. Below is a function named **initialize_network()** that creates a new neural network ready for
training. It accepts three parameters, the number of inputs, the number of neurons to have in the hidden layer and the number of outputs. You can see that for the hidden layer we create **n_hidden** neurons and each neuron in the
hidden layer has **n_inputs + 1** weights, one for each input column in a dataset and an additional one for the bias. You can also see that the output layer that connects to the hidden layer has **n_outputs** neurons, each with **n_hidden + 1** weights. This means that each neuron in the output layer connects to (has a weight for) each neuron in the hidden layer.

2. Forward Propagate

We can calculate an output from a neural network by propagating an input signal through each layer until the output layer outputs its values. We call this forward-propagation. It is the technique we will need to generate predictions during training that will need to be corrected, and it is the method we will need after the network is trained to make predictions on new data. We can break forward propagation down into three parts:

1. Neuron Activation.
2. Neuron Transfer.
3. Forward Propagation.

3. Back Propagate Error

The backpropagation algorithm is named for the way in which weights are trained. Error is calculated between the expected outputs and the outputs forward propagated from the network. These errors are then propagated backward through the network from the output layer to the hidden layer, assigning blame for the error and updating weights as they go. The math for backpropagating error is rooted in calculus, but we will remain high level in this section and focus on what is calculated and how rather than why the calculations take this
particular form.
This part is broken down into two sections.

1. Transfer Derivative.
2. Error Backpropagation.

4. Train Network

The network is trained using stochastic gradient descent. This involves multiple iterations of exposing a training dataset to the network and for each row of data forward propagating the inputs, backpropagating the error and updating the network weights. This part is broken down into two sections:

1. Update Weights.
2. Train Network.

5. Predict

Making predictions with a trained neural network is easy enough. We have already seen how to forward-propagate an input pattern to get an output. This is all we need to do to make a prediction. We can use the output values themselves directly as the probability of a pattern belonging to each output class. It may be more useful to turn this output back into a crisp class prediction. We can do this by selecting the class value with the larger probability. This is also called the arg max function. Below is a function named **predict ()** that implements this procedure. It returns the index in the network output that has the largest probability. It assumes that class values have been
converted to integers starting at 0.

Conclusion: -

Questions:

Q1. What is the feed-forward back propagation method?

Q2. What are the factors affecting back propagation network?

Q3. Which function is used on back propagation network?

Q4. What is the difference between backpropagation and feed forward?

Q5. What are the characteristics of backpropagation algorithm?

EXPERIMENT NO. 4

➢ **Aim:** Write a program to demonstrate ART
➢ **Outcome:** At end of this experiment, student will be able to illustrate ART neural network.
➢ **Hardware Requirement:**
➢ **Software Requirement:** Python IDE
➢ **Theory:**

Adaptive Resonance Theory (ART)

Adaptive resonance theory is a type of neural network technique developed by Stephen Grossberg and Gail Carpenter in 1987. The basic ART uses unsupervised learning technique. The term "adaptive" and "resonance" used in this suggests that they are open to new learning(i.e. adaptive) without discarding the previous or the old information(i.e. resonance). The ART networks are known to solve the stability-plasticity dilemma i.e., stability refers to their nature of memorizing the learning and plasticity refers to the fact that they are flexible to gain new information. Due to this the nature of ART they are always able to learn new input patterns without forgetting the past. ART networks implement a clustering algorithm. Input is presented to the network and the algorithm checks whether it fits into one of the already stored clusters. If it fits then the input is added to the cluster that matches the most else a new cluster is formed.

Types of Adaptive Resonance Theory(ART)

Carpenter and Grossberg developed different ART architectures as a result of 20 years of
research. The ARTs can be classified as follows:

ART1 – It is the simplest and the basic ART architecture. It is capable of clustering binary input values.

ART2 – It is extension of ART1 that is capable of clustering continuous-valued input data.

Fuzzy ART – It is the augmentation of fuzzy logic and ART.

ARTMAP – It is a supervised form of ART learning where one ART learns based on the previous ART module. It is also known as predictive ART.

FARTMAP – This is a supervised ART architecture with Fuzzy logic included.

Basic of Adaptive Resonance Theory (ART) Architecture

The adaptive resonant theory is a type of neural network that is self-organizing and competitive. It can be of both types, the unsupervised ones(ART1, ART2, ART3, etc) or the
supervised ones(ARTMAP). Generally, the supervised algorithms are named with the suffix
"MAP".

But the basic ART model is unsupervised in nature and consists of :

F1 layer or the comparison field(where the inputs are processed)

F2 layer or the recognition field (which consists of the clustering units)

The Reset Module (that acts as a control mechanism)

Department of AI & DS Engineering 2

The **F1 layer** accepts the inputs and performs some processing and transfers it to the F2 layer
that best matches with the classification factor.

There exist **two sets of weighted interconnection** for controlling the degree of similarity
between the units in the F1 and the F2 layer.

The **F2 layer** is a competitive layer. The cluster unit with the large net input becomes the
candidate to learn the input pattern first and the rest F2 units are ignored.
The **reset unit** makes the decision whether or not the cluster unit is allowed to learn the input
pattern depending on how similar its top-down weight vector is to the input vector and to the
decision. This is called the vigilance test.
Thus we can say that the **vigilance parameter** helps to incorporate new memories or new
information. Higher vigilance produces more detailed memories, lower vigilance produces
more general memories.
Generally **two types of learning** exists,slow learning and fast learning.In fast learning, weight update during resonance occurs rapidly. It is used in ART1.In slow learning, the weight change occurs slowly relative to the duration of the learning trial. It is used in ART2.

Advantage of Adaptive Resonance Theory (ART)

It exhibits stability and is not disturbed by a wide variety of inputs provided to its network.

It can be integrated and used with various other techniques to give more good results.

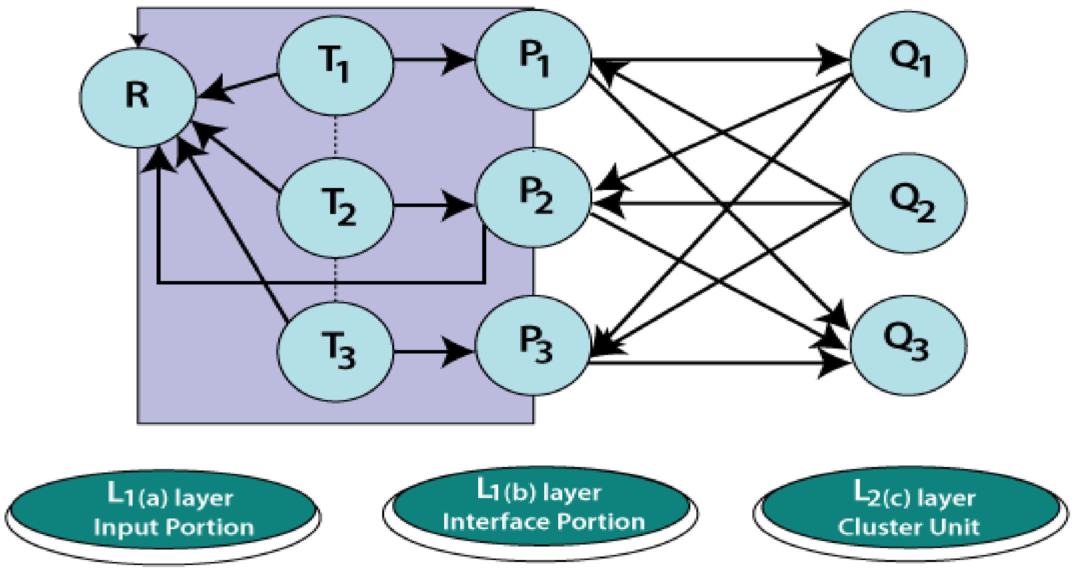
It can be used for various fields such as mobile robot control, face recognition, land cover classification, target recognition, medical diagnosis, signature verification, clustering web users, etc.

It has got advantages over competitive learning (like bpnn etc). The competitive learning lacks the capability to add new clusters when deemed necessary.

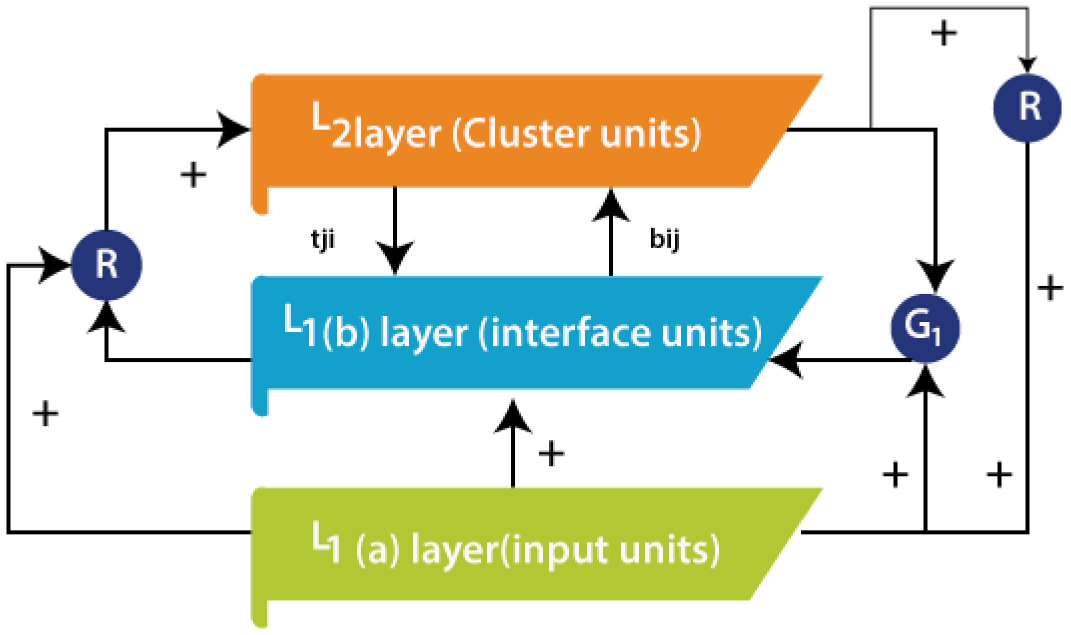
It does not guarantee stability in forming clusters.

Limitations of Adaptive Resonance Theory

Some ART networks are inconsistent (like the Fuzzy ART and ART1) as they depend upon
the order in which training data, or upon the learning rate.
ART1 is an unsupervised learning model primarily designed for recognizing binary patterns. It comprises an attentional subsystem, an orienting subsystem, a vigilance parameter, and a reset module, as given in the figure given below. The vigilance parameter has a huge effect on the system. High vigilance produces higher detailed memories. The ART1 attentional comprises of two competitive networks, comparison field layer L1 and the recognition field layer L2, two control gains, Gain1 and Gain2, and two short-term memory (STM) stages S1 and S2. Long term memory (LTM) follows somewhere in the range of S1 and S2 multiply the signal in these pathways.



Gains control empowers L1 and L2 to recognize the current stages of the running cycle. STM reset wave prevents active L2 cells when mismatches between bottom-up and top-down signals happen at L1. The comparison layer gets the binary external input passing it to the recognition layer liable for coordinating it to a classification category. This outcome is given back to the comparison layer to find out when the category coordinates the input vector. If there is a match, then a new input vector is read, and the cycle begins once again. If there is a mismatch, then the orienting system is in charge of preventing the previous category from getting a new category match in the recognition layer. The given two gains control the activity of the recognition and the comparison layer, respectively. The reset wave specifically and enduringly prevents active L2 cell until the current is stopped. The offset of the input pattern ends its processing L1 and triggers the offset of Gain2. Gain2 offset causes consistent decay of STM at L2 and thereby prepares L2 to encode the next input pattern without bais.



Conclusion: -

Questions:

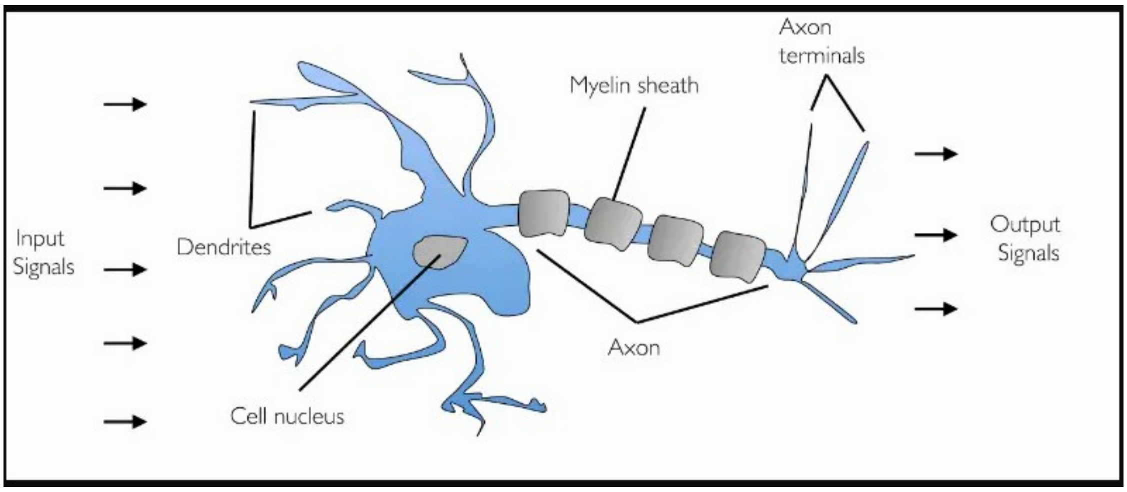
Q1. Explain adaptive resonance learning with its applications ? Q2.
Explain task of feedforward ANN in pattern recognition ?
Q3. What are advantage of disadvantage of simulated annealing ?
Q4. Discuss use of hopfield networks in associative learning ?
Q5. What is the purpose of ART networks?

EXPERIMENT NO. 5

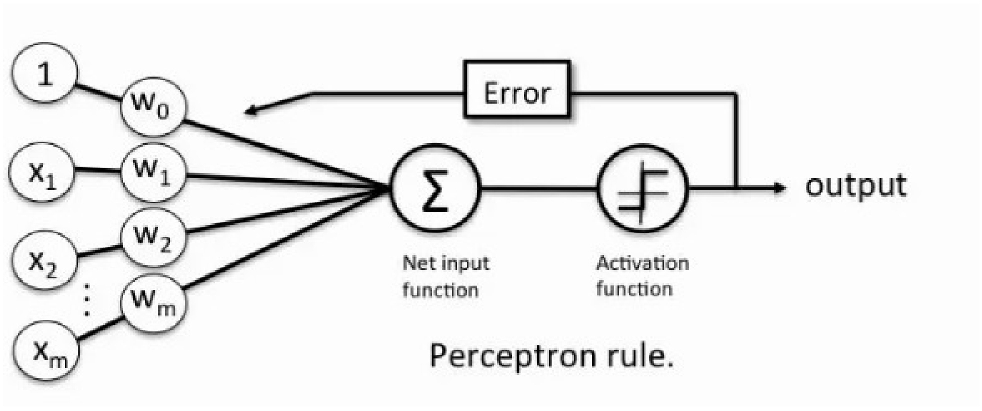
- **Aim:** Write a program to demonstrate the perceptron learning law with its decision region using python. Give the output in graphical form
- **Outcome:** At the end of this experiment, students will be able to demonstrate the perceptron learning law with its decision regions using python.
- **Software Requirement:** Ubuntu OS,Python Editor(Pyhton Interpreter)
- **Theory:**

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

flows:



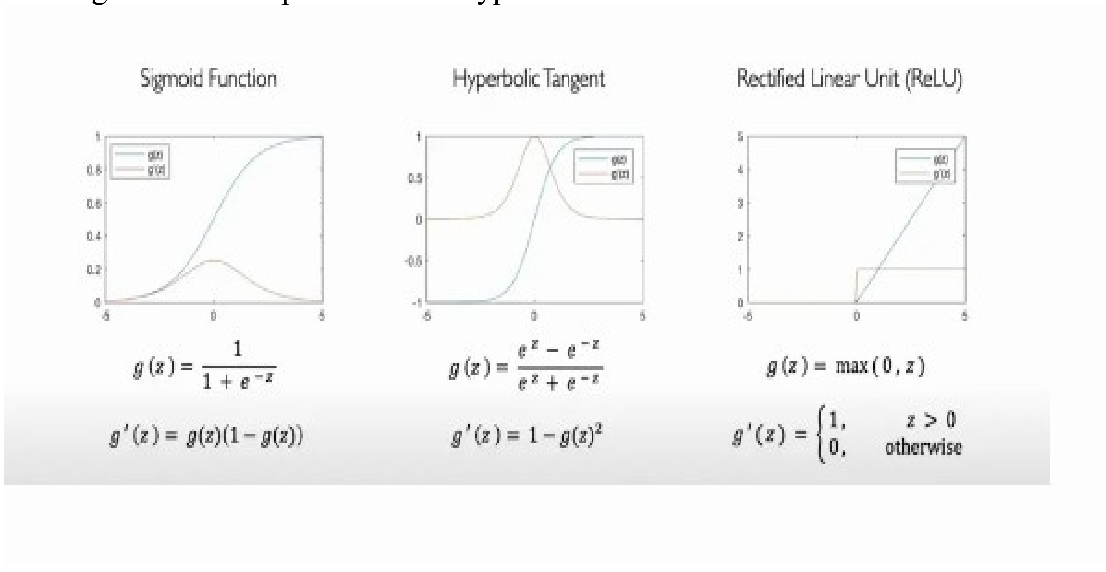
The perceptron when represented as line diagram would look like the following with mathematical notations:



Pay attention to some of the following in relation to what's shown in the above diagram representing a neuron:

- **Step 1 – Input signals weighted and combined as net input**: Weighted sums of input signal reaches to the neuron cell through dendrites. The weighted inputs does represent the fact that different input signal may have different strength, and thus, weighted sum. This weighted sum can as well be termed as **net input** to the neuron cell.
- **Step 2 – Net input fed into activation function:** Weighted The weighted sum of inputs or **net input** is fed as input to what is called as **activation function**. The activation function is a non-linear activation function. The activation functions are of different types such as the following:
  - Unit step function
  - Sigmoid function (Popular one as it outputs number between 0 and 1 and thus can be used to represent probability)
  - Rectilinear (ReLU) function
  - Hyperbolic tangent

  The diagram below depicts different types of non-linear activation functions.



- **Step 3A – Activation function outputs binary signal appropriately**: The activation function processes the net input based on the **unit step** (Heaviside**) function** and outputs the binary signal appropriately as either 1 or 0. The activation function for perceptron can be said to be a unit step function. Recall that the **unit step function**, u(t), outputs the value of 1 when t >= 0 and 0 otherwise. In the case of a shifted unit step function, the function u(t-a) outputs the value of 1 when t >= a and 0 otherwise.
- Step 3B – Learning input signal weights based on prediction vs actuals: A parallel step is

a neuron sending the feedback to strengthen the input signal strength (weights) appropriately such that it could create an output signal appropriately that matches the actual value. The feedback is based on the outcome of the activation function which is a unit step function. Weights are updated based on the **gradient descent learning algorithm.** Here is my post on gradient descent – **Gradient descent explained simply with examples**. Here is the equation based on which the weights get updated:

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

where

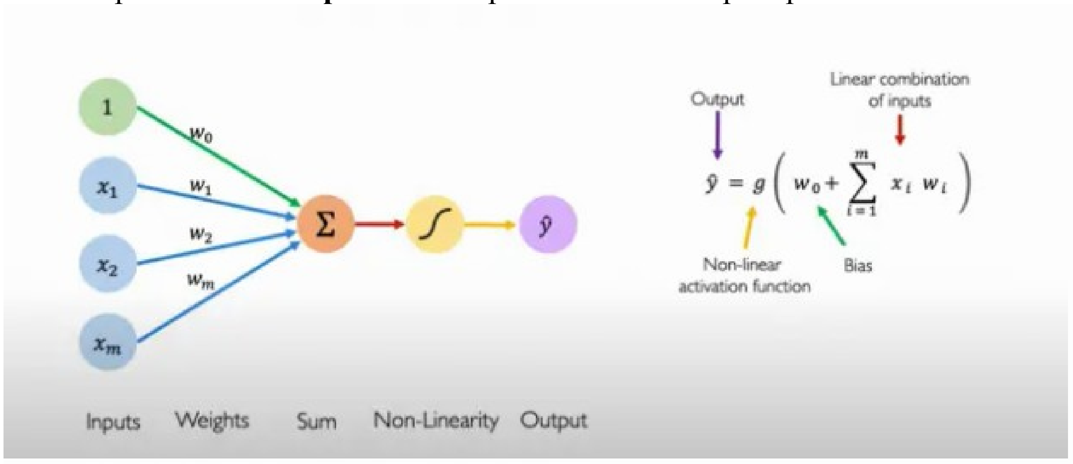
$$\Delta w_i = \underset{\substack{\text{learning}\\ \text{rate}}}{\eta}(\underset{\substack{\text{target}\\ \text{value}}}{t} - \underset{\substack{\text{perceptron}\\ \text{output}}}{o})\underset{\substack{\text{input}\\ \text{value}}}{x_i}$$

Pay attention to some of the following in above equation vis-a-vis Perceptron learning algorithm:

- Weights get updated by δw ��

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

Perceptron – A single-layer neural network comprising of a single neuron
Here is another picture of **Perceptron** that represents the concept explained above.


1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
$\hat{y}$
Output
Linear combination of inputs
$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$
Non-linear activation function
Bias
Inputs
Weights
Sum
Non-Linearity
Output

Conclusion:

-

____

____

____

____

____

Questions:

Q1. What do you mean by Perceptron?

Q2. What are the different types of Perceptrons?

Q3. What is the use of the Loss functions?