Data Mining

Lecture 12

Ananya Jana
CS360

Fall 2024

Acknowledgement

Some of the slides have been taken from the presentation created by the authors of this courses's textbook "Introduction to Data Mining".

Artificial Neural Network

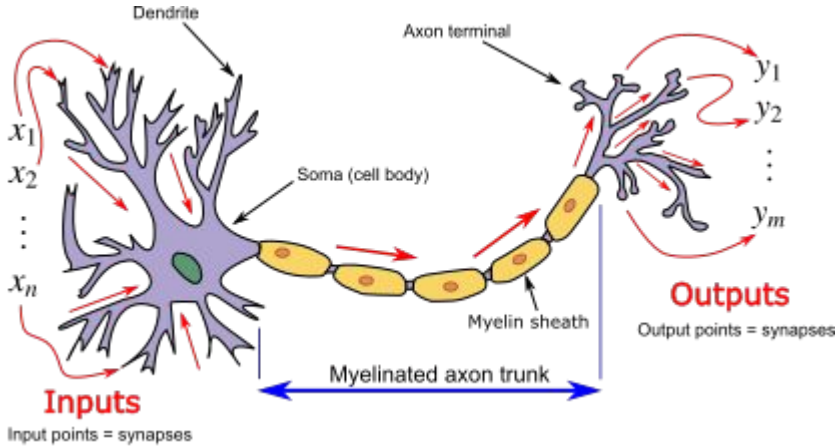
A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain.

source:https://aws.amazon.com/what-is/neural-network/

Artificial neural networks (**ANNs**, also shortened to neural networks (NNs) or *neural nets*) are a branch of machine learning models that are built using principles of neuronal organization discovered by connectionism in the biological neural networks constituting animal brains.[1][2]

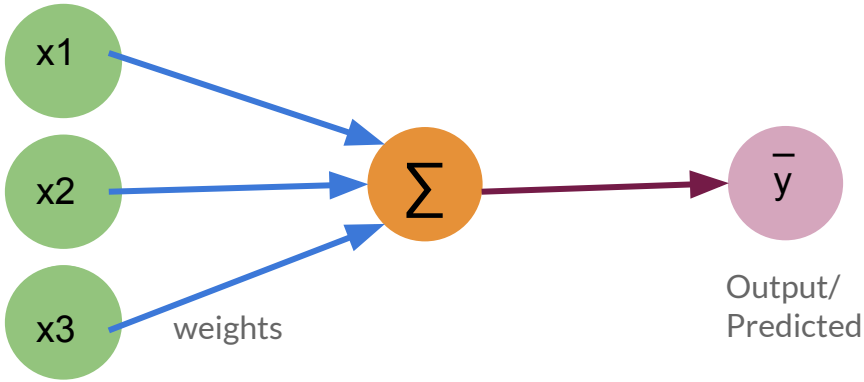
Source: wiki

Artificial Neural Network



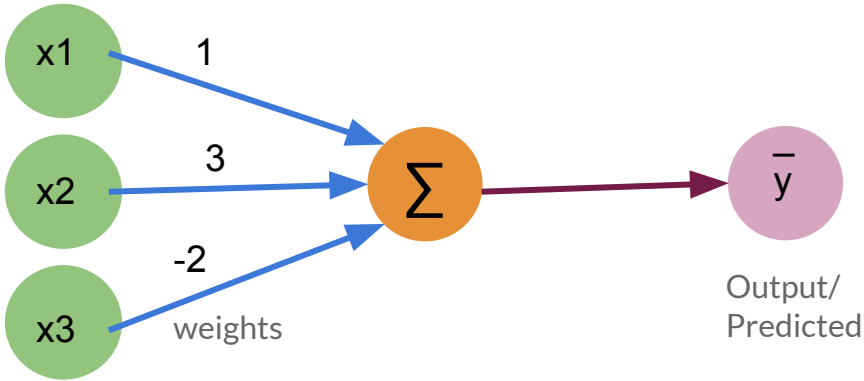
Neurons in human

Artificial Neural Network



x1	x2	x3	y(Actual)
5	6	7	10

Artificial Neural Network



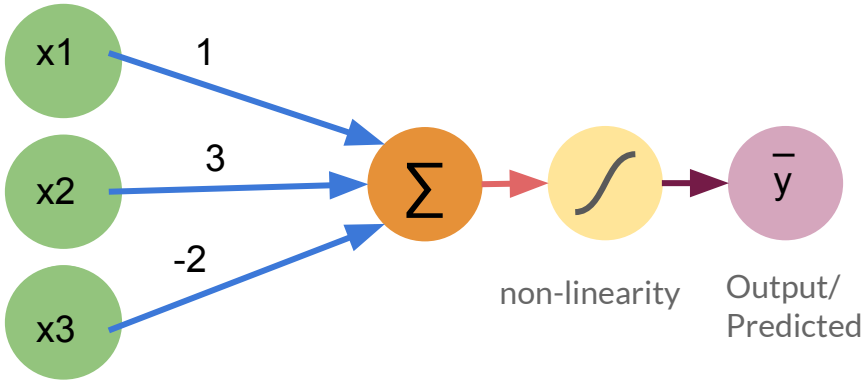
x1	x2	x3	y(Actual)
5	6	7	10

$\bar{y} = 1*5 + 3*6 + (-2)*7 = 9$

Error = predicted value - Actual Value
= 9 - 10
= -1

$\bar{y} = 1*x1 + 3*x2 + (-2)*x3$

Artificial Neural Network



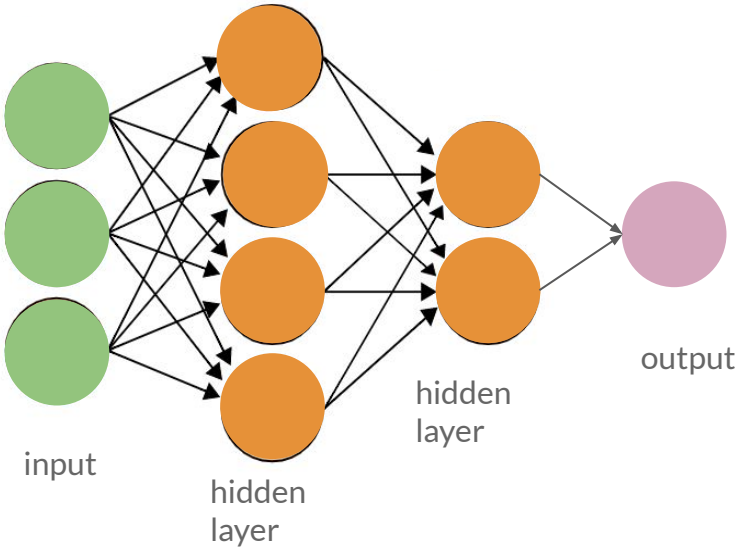
x1	x2	x3	y(Actual)
5	6	7	10

$\bar{y} = g(1*x1 + 3*x2 + (-2)*x3)$

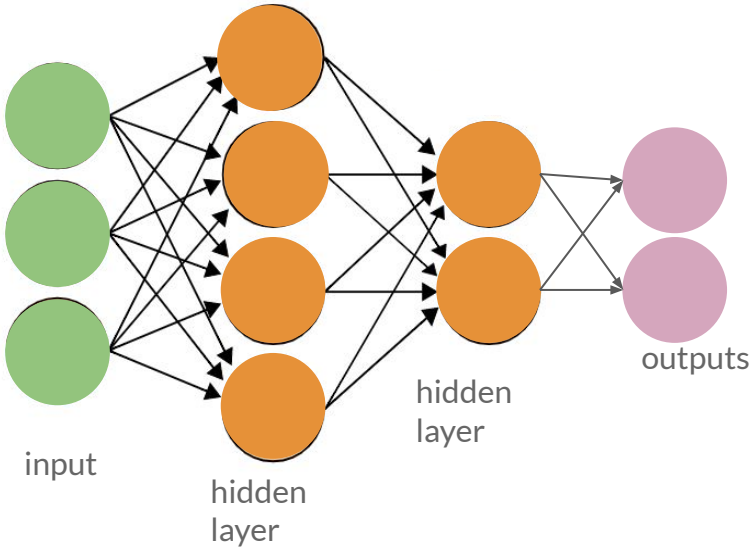
$\bar{y} = g(1*5 + 3*6 + (-2)*7) = g(9)$

Backpropagate the error

Artificial Neural Network



Artificial neural network with
multiple hidden layers



Artificial neural network with
multiple hidden layers and multiple outputs

Activation Function

Activation function of a node in an artificial neural network is a function that calculates the output of the node (based on its inputs and the weights on individual inputs)

Usually it maps the output values to a range e.g. 0 to 1 or -1 to 1 etc.

Activation functions can be both linear and non-linear
Usually non-linear activation functions are used in neural networks.

Source: wiki

Different Activation functions used in Artificial Neural Network

Sigmoid function:

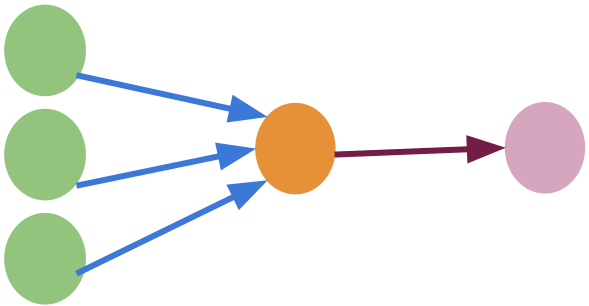
$$\frac{1}{1+e^{-x}}$$

Tanh function:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Feed forward Neural Network

The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist.

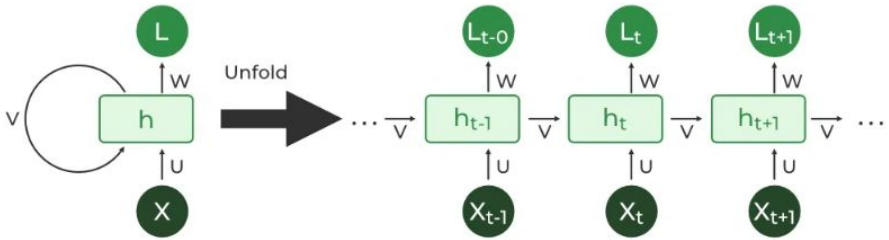


data/input travels forward

Text Source: geeksforgeeks

Recurrent Neural Network

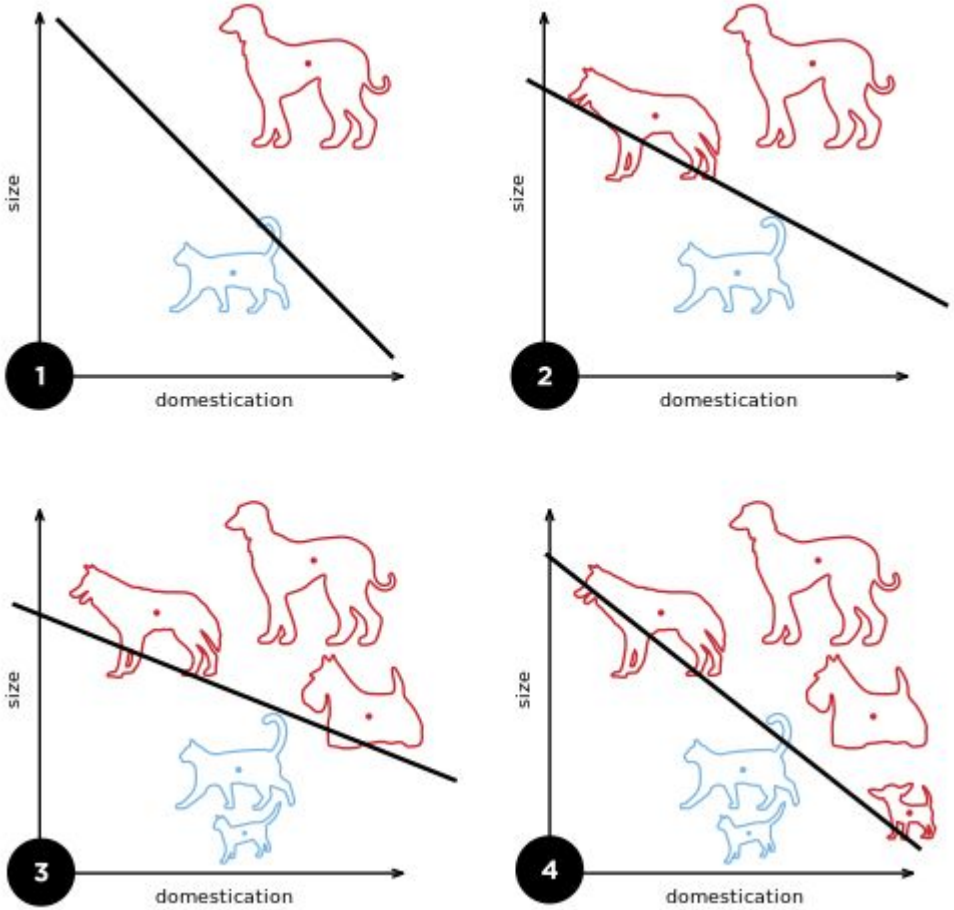
The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computation



output from the previous step is fed as input to the current step

Source: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/

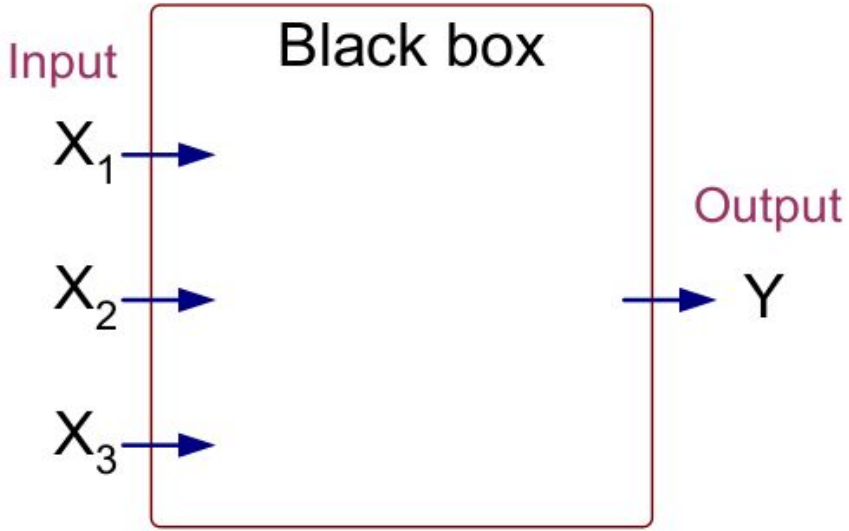
Perceptron



Perceptron Decision Boundary changes as more examples are included(image source wiki)

Artificial Neural Network (ANN)

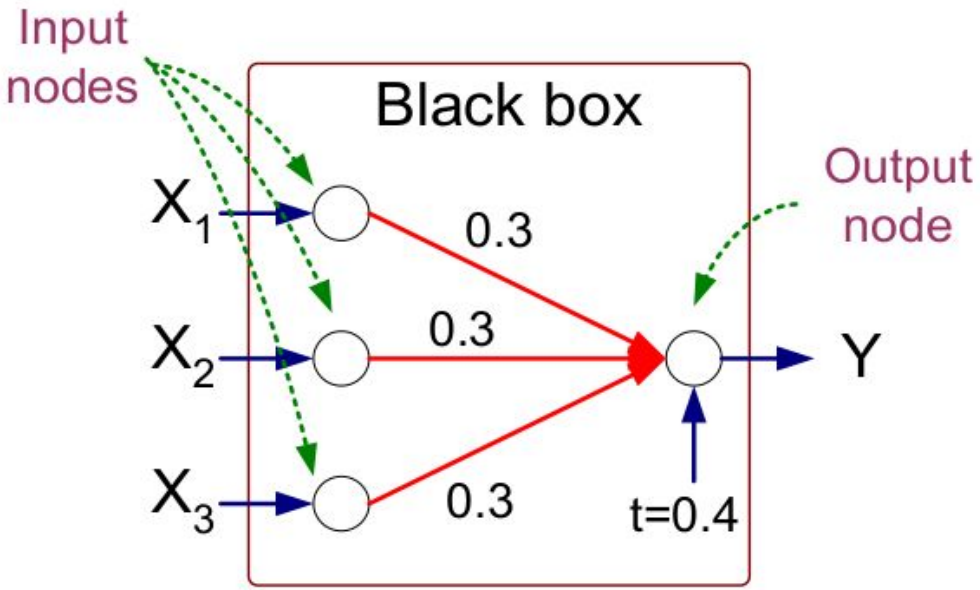
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Network (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

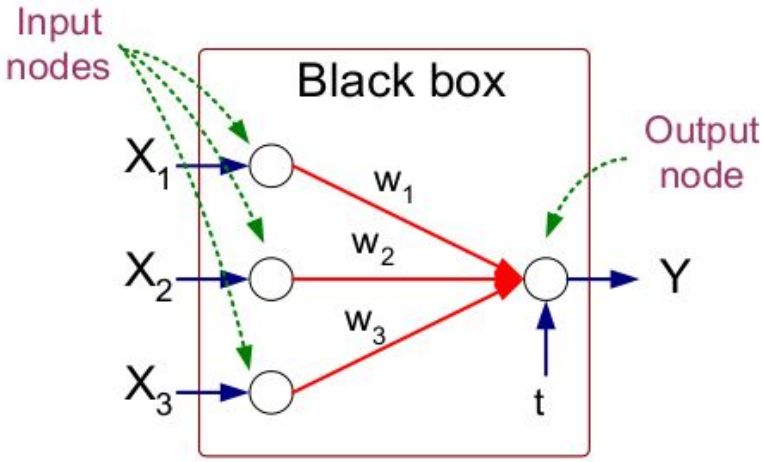


$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Network (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t

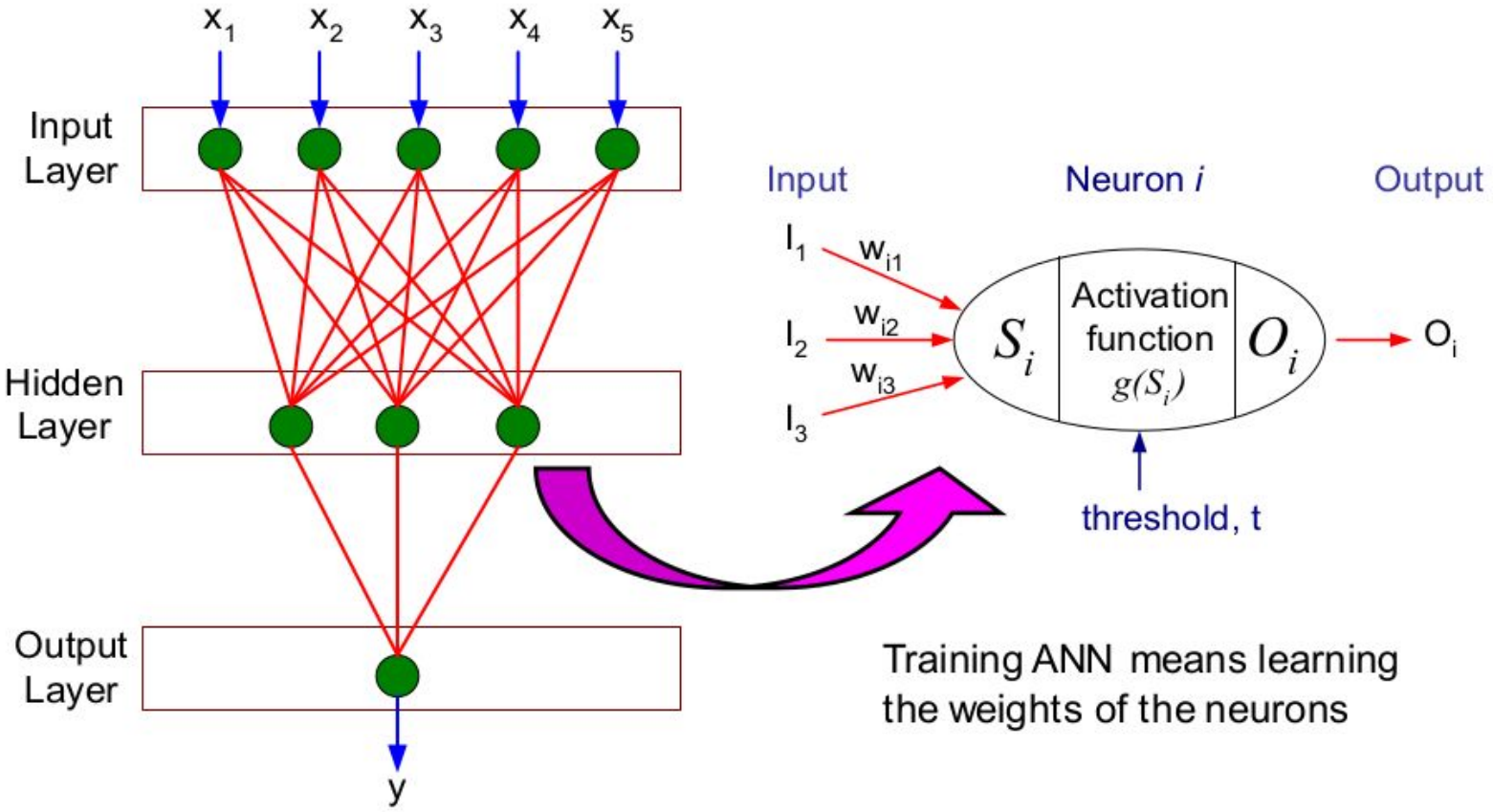


Perceptron Model

$$Y = sign(\sum_{i=1}^{d} w_i X_i - t)$$

$$= sign(\sum_{i=0}^{d} w_i X_i)$$

General structure of ANN

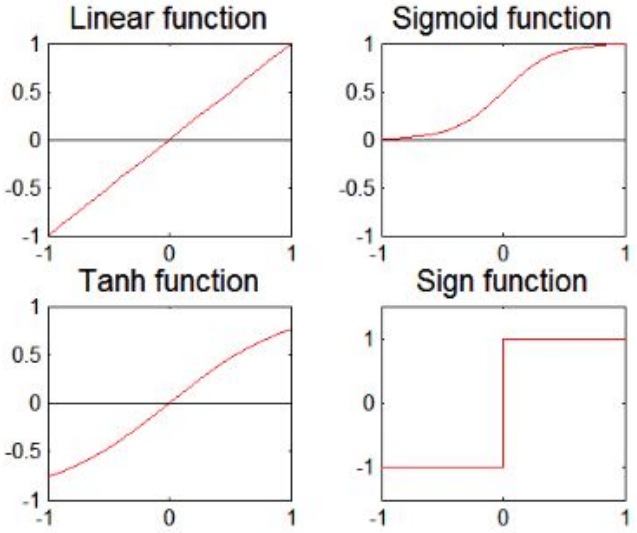


Artificial Neural Network (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f\left(\sum_i w_i X_i\right)$$



Perceptron

- Single layer network
  - Contains only input and output nodes
- Activation function: f = sign(w•x)
- Applying model is straightforward

$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

  - $X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = sign(0.2) = 1$

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute f(w, x_i)
    - Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

- Until stopping condition is met

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i \;;\; \lambda : \text{learning rate}$$

- Intuition:
  - Update weight based on error: $e = \left[y_i - f(w^{(k)}, x_i)\right]$
  - If y=f(x,w), e=0: no update needed
  - If y>f(x,w), e=2: weight must be increased so that f(x,w) will increase
  - If y<f(x,w), e=-2: weight must be decreased so that f(x,w) will decrease

Example of Perceptron Learning

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

$$Y = sign(\sum_{i=0}^{d} w_i X_i)$$

$$\lambda = 0.1$$

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

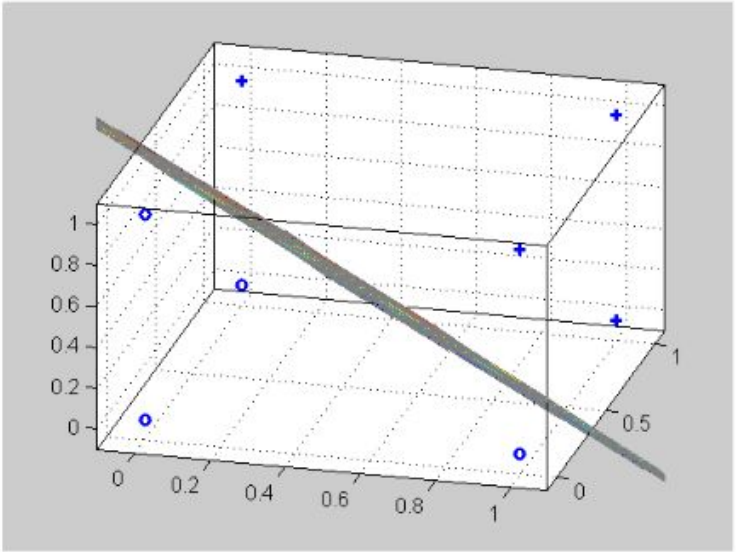
Task

1. Suppose you have the following table where X1 and X2 are the attributes and Y is the label. Our goal is to build a perceptron model on this dataset. Please write down the different steps of building the perceptron model building for this dataset. Show the weight calculation after passing each data point through the model. (You can choose the initial weights randomly. You can choose to pass the data points in any order)

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

Perceptron Learning Rule

- Since f(w,x) is a linear combination of input variables, decision boundary is linear



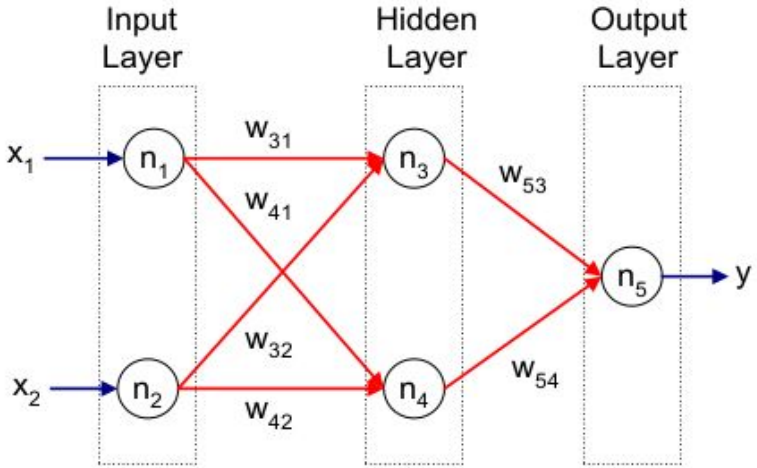
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

Multilayer Neural Network

- Hidden layers
  - intermediary layers between input & output layers
- More general activation functions (sigmoid, linear, etc)

Multilayer Neural Network

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



Learning Multilayer Neural Network

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term e = y-f(w,x) and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Gradient Descent for Multilayer NN

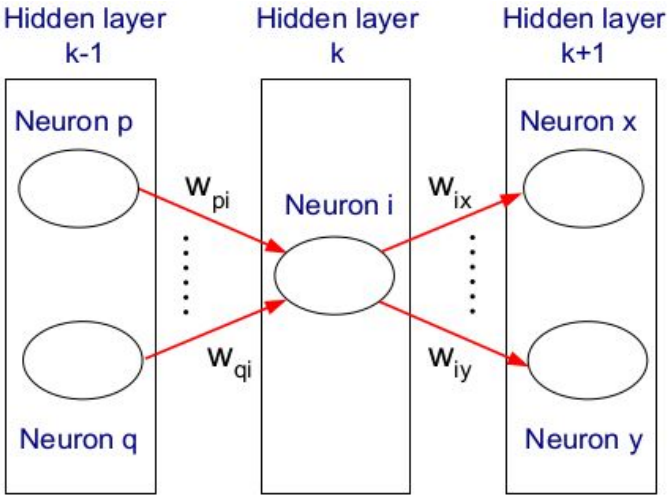
- Weight update: $w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$
- Error function: $E = \frac{1}{2} \sum_{i=1}^{N} \left(t_i - f(\sum_j w_j x_{ij}) \right)$
- Activation function f must be differentiable
- For sigmoid function:

$$w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i (t_i - o_i) o_i (1 - o_i) x_{ij}$$

- Stochastic gradient descent (update the weight immediately)

Gradient Descent for Multilayer NN

- For output neurons, weight update formula is the same as before (gradient descent for perceptron)



- For hidden neurons:

$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i (1 - o_i) \sum_{j \in \Phi_i} \delta_j w_{ij} x_{pi}$$

$$\text{Output neurons} : \delta_j = o_j (1 - o_j)(t_j - o_j)$$

$$\text{Hidden neurons} : \delta_j = o_j (1 - o_j) \sum_{k \in \Phi_j} \delta_k w_{jk}$$

Design Issues in ANN

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - k or $\log_2$ k nodes for each categorical attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $\log_2$ k nodes for k-class problem
- Number of nodes in hidden layer
- Initial weights and biases

Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large
- Gradient descent may converge to local minimum
- Model building can be very time consuming, but testing can be very fast
- Can handle redundant attributes because weights are automatically learnt
- Sensitive to noise in training data
- Difficult to handle missing attributes