classmate
Date
Page

UNIT 1:

Introduction to artificial neural networks

ANN

- Artificial Neural Network (ANN) is commonly referred as Neural Network (NN). It is the computational paradigm that is motivated from the way the computation is performed by human brain or nervous system.

- Brain is a highly complex, non-linear, and parallel computation system that can perform computations like perception, pattern recognition, motor control, etc. Neuron or nerve cell is the basic structural unit of brain.

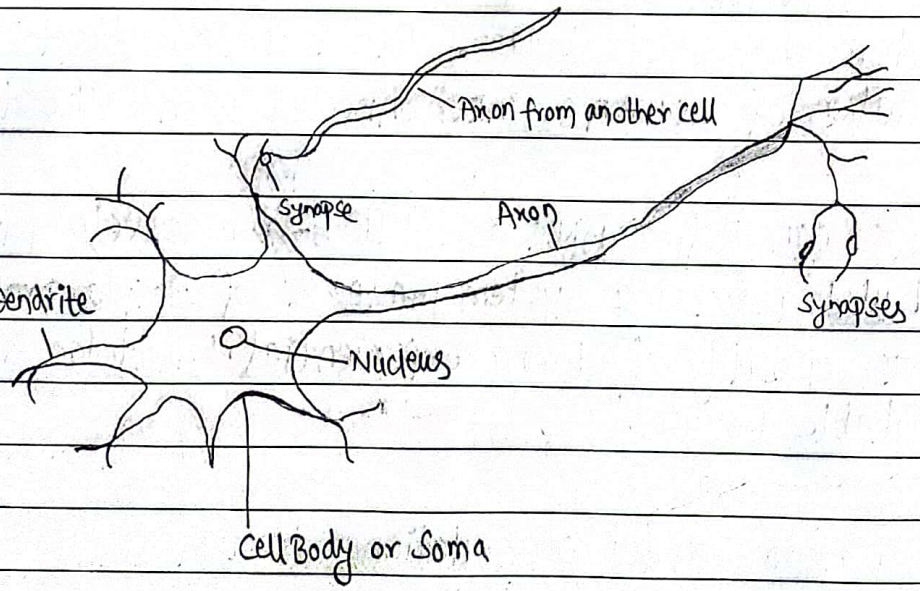
- Human can perform the task much faster than the fastest digital computers that exists todays. This is possible due to parallel computation of neurons interconnected with each other.

- Thus we can define ANN as "It is a massively parallel distributed processing system made up of simple processing units that has capability of storing experiential knowledge and making it available for use."

- ANNs perform useful computations through the process of learning by using some algorithm.

Biological Neural Networks

- Most living creatures, which have the ability to adapt to a changing environment, need a controlling unit which is able to learn.
- Humans have very complex networks of highly specialized neurons to perform this task.
- Human brain consists of a very large number of neurons, about 10^{11} in average. These can be seen as the basic building bricks for the central nervous system.
- The neurons are interconnected at points called synapses. The complexity of the brain is due to the massive number of highly interconnected simple units working in parallel, with an individual neuron receiving input from up to 10000 others.



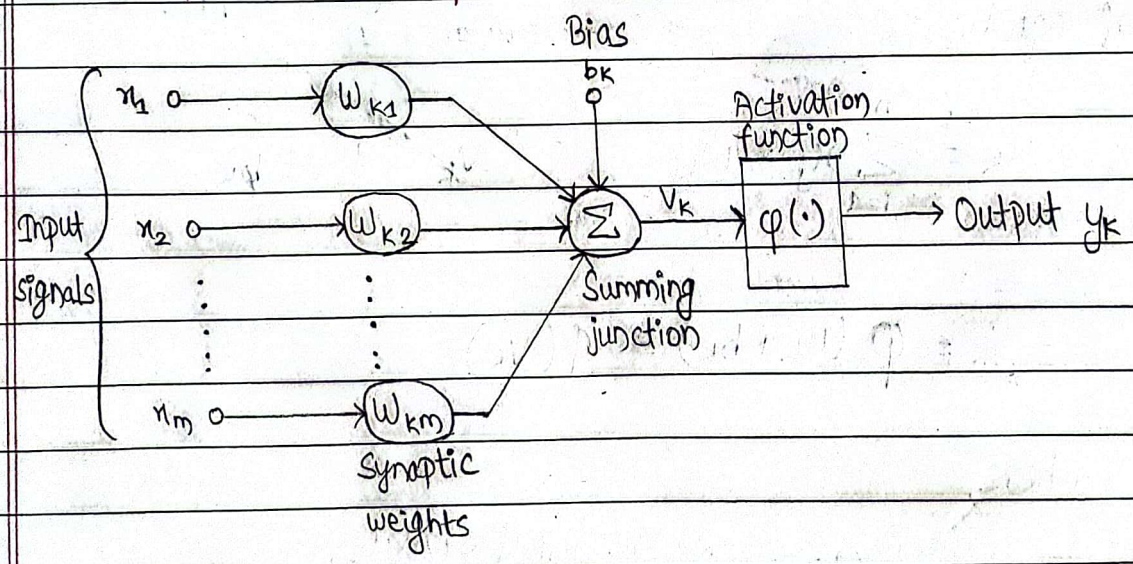
- A human neuron contains:
  - a cell body for signal processing,
  - many dendrites to receive signals,
  - an axon for outputing the result, and
  - synapses between the axon and dendrites of other cells.

- Signals come into the dendrites through the synapses of other neurons.
- All signals from all dendrites are summed up in the cell body.
- When the sum is larger than a threshold, the neuron fires, and sends out an output signal to other neurons through the axon.
- The end of the axon is divided in many branches, called synapses, which are then connected to other dendrites of other cells.

Models of Neuron

- A neuron is an information-processing unit that is fundamental to the operation of a neural network.
- Basically, Models of neuron can be divided into two categories:
  - Deterministic model of Neuron
  - Stochastic model of Neuron

1. Deterministic Model of Neuron



Three basic elements of this neural model are:

- Synapses or connecting links
- Adder
- Activation Function

- Synapses: These are the connecting links that are used to collect input for the neuron. Each link is characterized by weight that defines strength of the link.

- Adder: It is responsible for finding weighted sum of inputs to the neuron.

- Activation Function: It is responsible for finding output of the neuron. It is also referred as squashing function.

- The neural model also includes an externally applied bias, denoted by b_k. The bias b_k has the effect of increasing or lowering the net input. In mathematical terms, we may describe the neural model by writing the set of equations:

$$u_k = \sum_{j=1}^{n} x_j * w_{kj} \qquad v_k = u_k + b_k$$

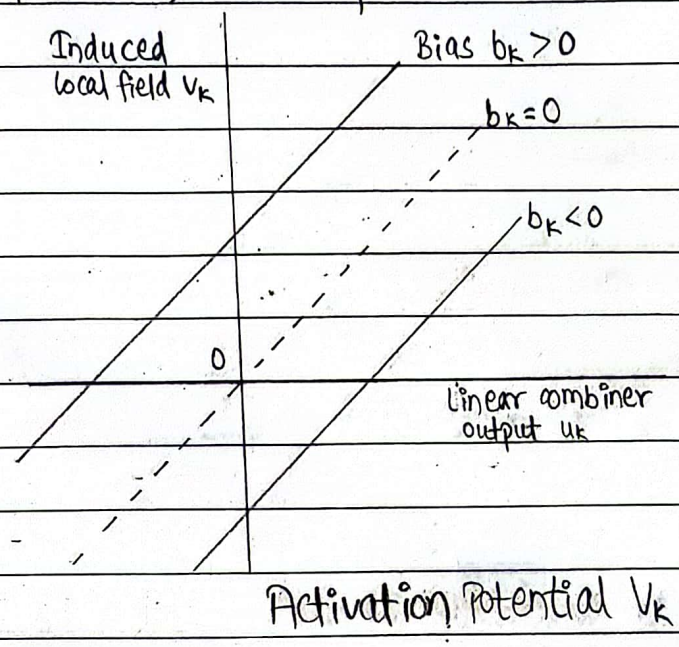
$$y_k = \varphi(u_k + b_k) = \varphi(v_k)$$

where,

$x_1, x_2, \ldots, x_n$ are input signals,
$w_{k1}, w_{k2}, \ldots, w_{kn}$ are weights,
u_k is weighted sum of inputs,
φ is activation function, and
y_k is output signal.

- The use of bias b_k has the effect of applying an affine transformation to the output u_k of the linear combiner in the neural model.

- Depending on whether the bias b_k is positive or negative, the relationship between the activation potential (v_k) and the linear combiner output (u_k) is modified as below:

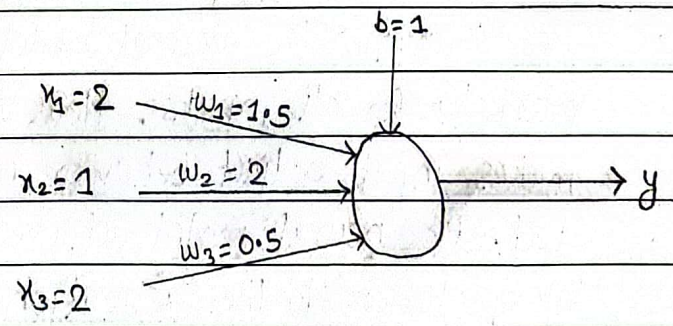


Example: Consider following neuron and compute its output by assume activation function $F(x) = 1$ if $x > 5$ and $F(x) = 0$, otherwise

$u = x_1 * w_1 + x_2 * w_2 + x_3 * w_3$

$= 2 * 1.5 + 1 * 2 + 2 * 0.5$

$= 6$



$v = u + b$

$= 6 + 1$

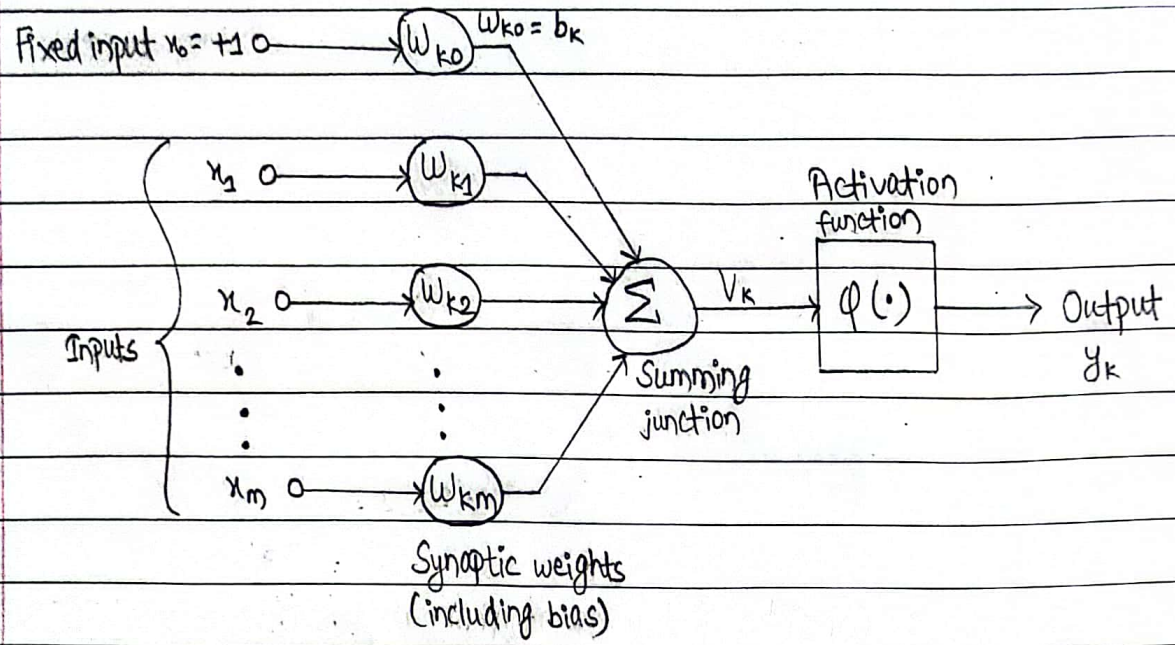
$= 7$

Now,

$y = f(v) = 1$

- We can reformulate the model of neuron by doing two things:
  1. Adding a new input signal fixed at 1, and
  2. Adding a new synaptic weight equal to the bias b_k.

- Although the two models are different in appearance, they are mathematically equivalent.



2. Stochastic Model of Neuron

- The deterministic neural model defines input-output behavior precisely for all inputs. However, stochastic model of neuron makes input-output behavior non-deterministic.
- Stochastic neural model achieves this by giving probabilistic interpretation to the activation function used in deterministic neural model.
- Specifically, a neuron is permitted to reside in only one of two states: +1 (ON) or -1 (OFF). The decision for a neuron to fire is probabilistic. Let x denote the state of the neuron and $P(v)$ denote the probability of firing, where v is the activation potential of the neuron.

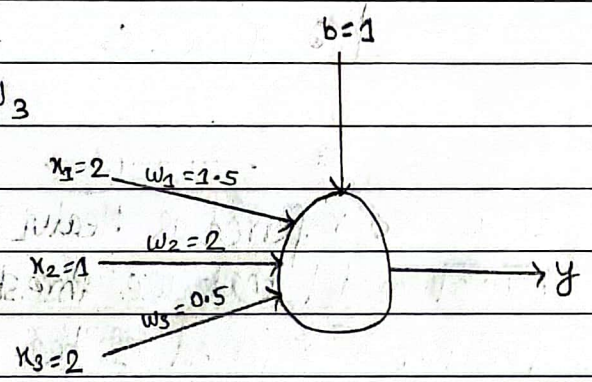
$$x = \begin{cases} +1 & \text{with probability } p(v) \\ -1 & \text{with probability } 1-p(v) \end{cases}$$

$$p(v) = \frac{1}{1+e^{-v/T}},$$ where T is parameter to control noise level

- This adds uncertainty to firing of neuron and hence makes the input-output behavior stochastic. Rest of things in stochastic model of neuron is similar to the deterministic model.

Example:

Consider following stochastic neuron and compute its probability of firing by assuming $T = 5$.



$$u = x_1 * w_1 + x_2 * w_2 + x_3 * w_3$$
$$= 2*1.5 + 1*2 + 2*0.5$$
$$= 6$$

$$v = u + b$$
$$= 6 + 1$$
$$= 7$$

Now,

$$P(v) = \frac{1}{(1+e^{-v/T})}$$
$$= \frac{1}{(1+e^{-7/5})}$$
$$= 0.802$$

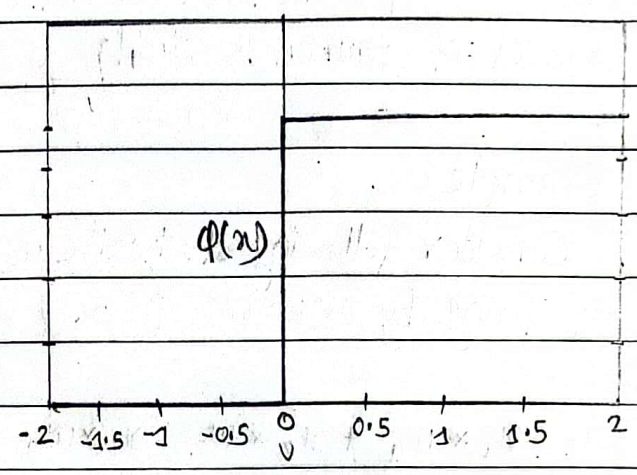
Thus, the probability of firing the neuron is 0.802.

Activation Functions

- Activation functions are the functions responsible to convert a input signal of a node in a ANN to an output signal.
- The activation function is the non-linear transformation that we do over the input signal. This transformed output is then sent to the next layer of neurons as input.
- Some widely used activation functions are: Threshold, linear, sigmoid, tanh, etc.

Threshold Function

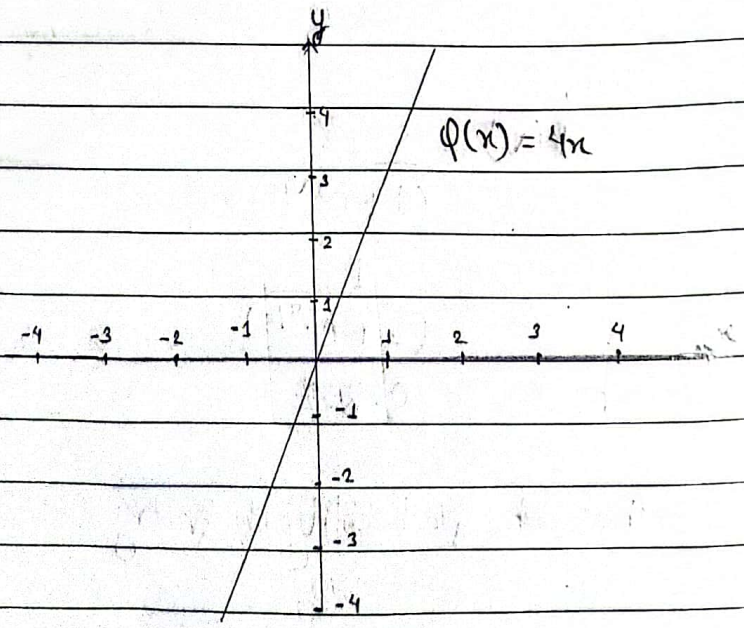
$$\varphi(x) = \begin{cases} 1 & \text{if } x \geq c \\ 0 & \text{if } x < c \end{cases}$$



- It is also referred as Heaviside function. The non-linear neural model that use threshold function as activation function is referred as the McCulloch-Pitts model.

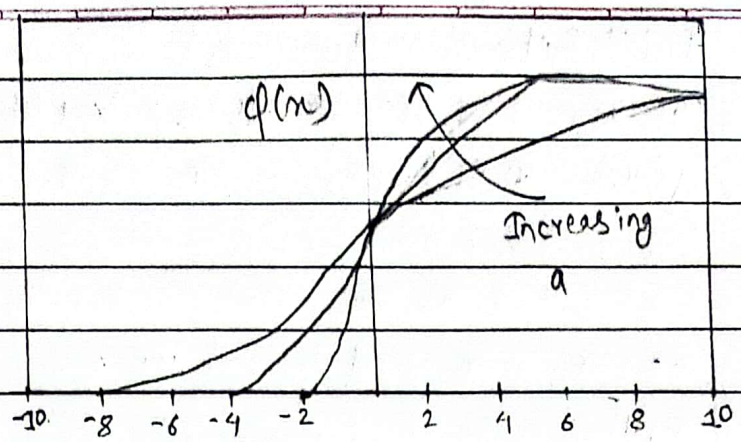
Linear Function

$f(x) = ax + b$



Sigmoid Function

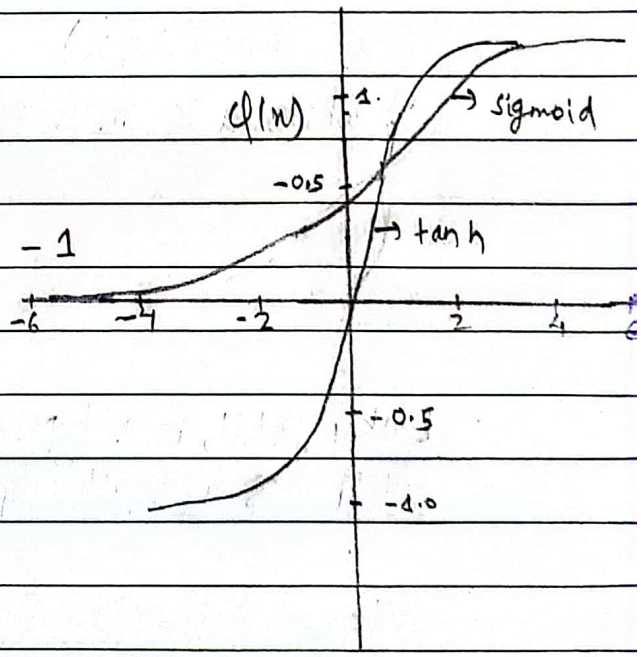
$$\varphi(x) = \frac{1}{1+e^{-ax}}$$



- By varying the parameter a (slope), we obtain sigmoid functions of different slopes. The sigmoid function is the class of functions whose graph is S-shaped curve. It is the most common form of activation function used in the construction of neural networks. An example of the sigmoid function is the logistic function, where $a = 1$. It squashes the output in the range $(0, 1)$.

Tanh Function

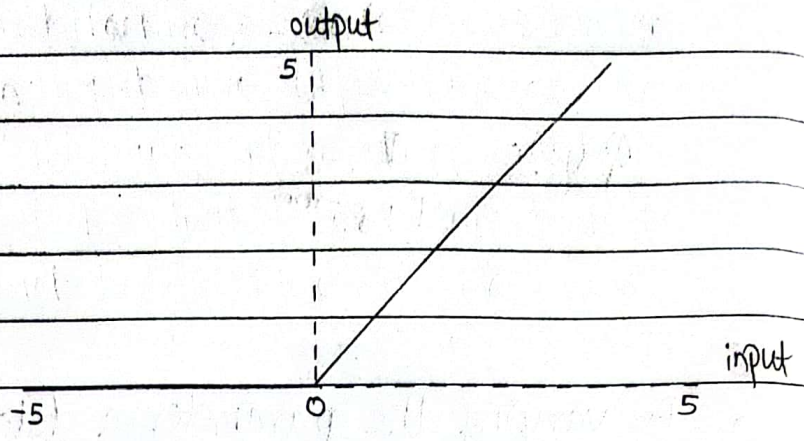
$$\varphi(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{2}{1+e^{-2x}} - 1$$



- It has characteristics similar to sigmoid that we discussed above. But, it squashes the output between $(-1, 1)$. Tanh is also a very popular and widely used activation function. It is special case of sigmoid function

ReLU Activation Function

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

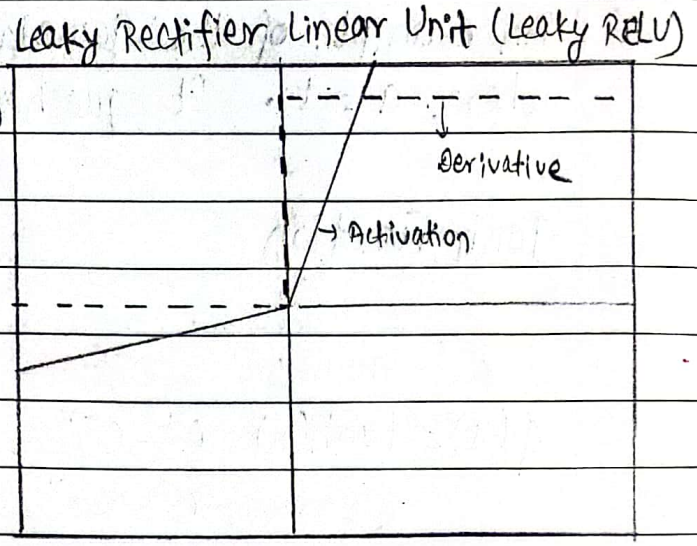


- ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. This function is also non-linear.

Leaky RELU Activation Function

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

where, a is small value



- Leaky ReLU, is a type of activation function based on a RELU, but it has a small slope for negative values instead of a flat slope.

Softmax Activation Function

$$S(x) = \frac{e^{x}}{\sum_{i=1}^{n} e^{x_i}} \quad \text{for } i = 1, 2, 3, \ldots, n.$$

$$\begin{bmatrix} 1.2 \\ 0.9 \\ 0.4 \end{bmatrix} \rightarrow \text{Softmax} \rightarrow \begin{bmatrix} 0.46 \\ 0.34 \\ 0.20 \end{bmatrix}$$

- Softmax is fundamentally a vector function. It takes a vector as input and produces a vector as output.

- The softmax function also squashes the outputs of each unit to be between 0 and 1. But it also divides each output such that the total sum of the outputs is equal to 1. The output of the softmax function tells you the probability that any of the classes are true.

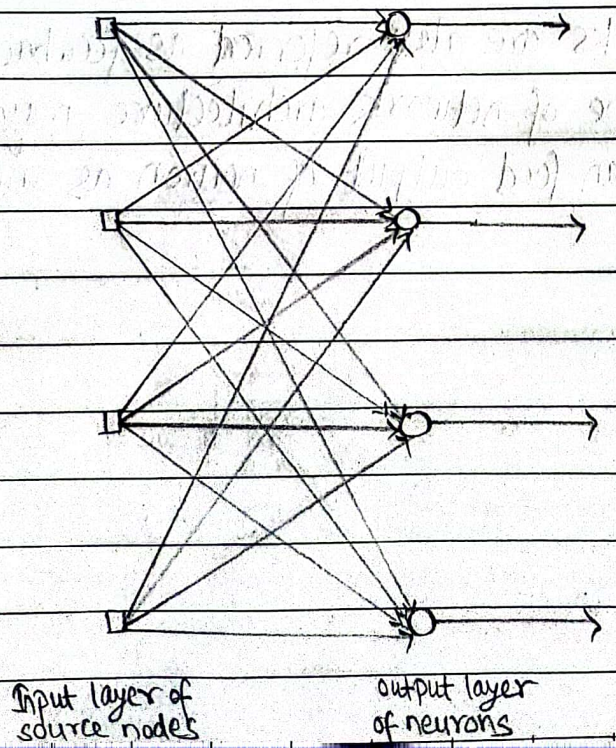
Structures of Neural Network

The manner in which neurons of a neural network are structured is called neural network architecture. Broadly, we can divide neural network architectures or structures into three categories.

- Single-Layer Feedforward Networks
- Multi-Layer Feedforward Networks
- Recurrent Networks

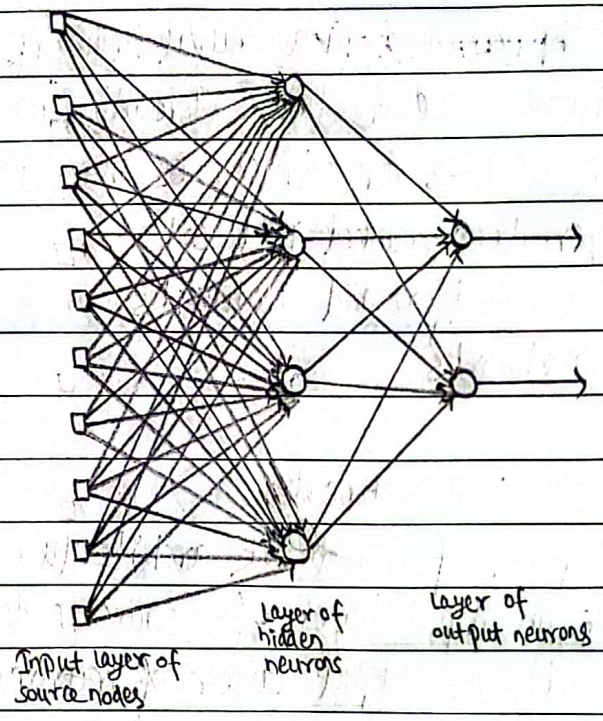
Single-Layer Feedforward Networks

- It is the simplest form of a network architecture. In this architecture we have an input layer of source nodes that are connected directly with an output layer of neurons (computation nodes), but not vice-versa.



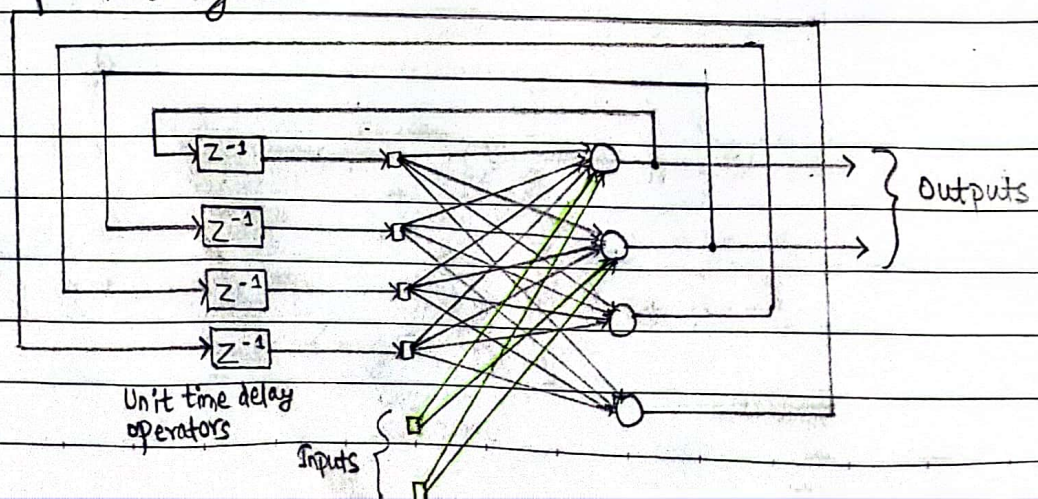
Multi-Layer Feedforward Networks

- In this type of network architecture, one or more hidden layers are present between input and output layers. These layers are not directly visible and information only flows in the direction of input to output layer. This type of network is designed to extract higher order statistics from input.



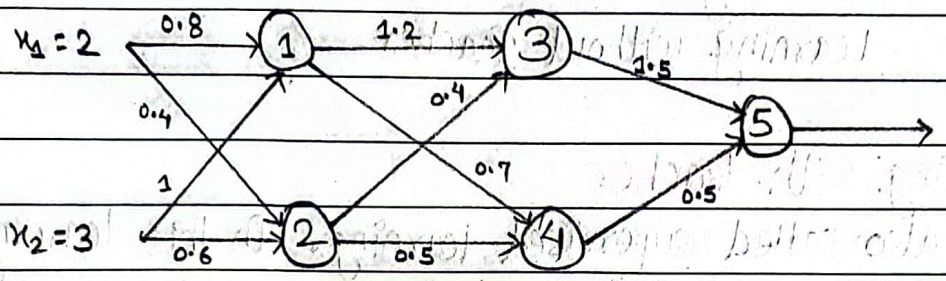
Recurrent Networks

- This class of networks are also referred as feedback neural networks. This type of network architecture may contain feedback loop that can feed output of neuron as input to neurons of previous layers.



Example:

Consider following Neural Network and compute its output using activation function $F(x) = 2x - 1$. Weights of synaptic links are provided above each link.



For Node 1

$u_1 = 2*0.8 + 3*1 = 4.6 \Rightarrow y_1 = f(u_1) = 2*4.6 - 1 = 8.2$

For Node 2

$u_2 = 2*0.4 + 3*0.6 = 2.6 \Rightarrow y_2 = f(u_2) = 2*2.6 - 1 = 4.2$

For Node 3

$u_3 = 8.2*1.2 + 4.2*0.4 = 11.51 \Rightarrow y_3 = f(u_3) = 22.04$

For Node 4

$u_4 = 8.2*0.7 + 4.2*0.5 = 7.84 \Rightarrow y_4 = f(u_4) = 14.68$

For Node 5

$u_5 = 22.04*1.5 + 14.68*0.5 = 40.4 \Rightarrow y_5 = f(u_5) = 79.8$

Thus, final output of the neural network $(y) = 79.8$.

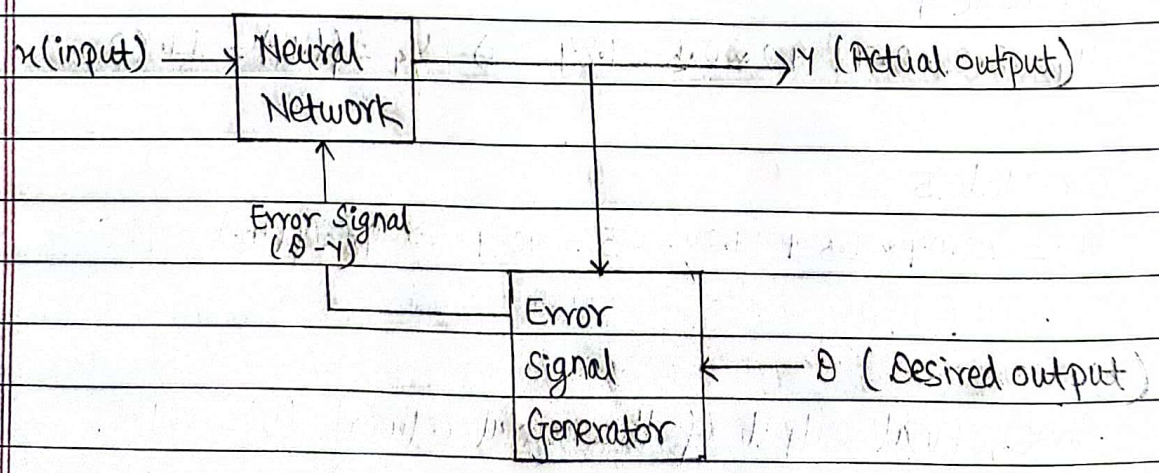
Learning Principles

Neural networks learns from its environment. Broadly, we can categorize learning principles in neural networks into two categories.

- Learning with Teacher
- Learning without Teacher

Learning with Teacher

- It is also called supervised learning. In this learning paradigm, we present examples of correct input - output pairs to the neural network during the training phase.
- This training set of examples is equivalent to the teacher for the neural network. During the training of ANN under supervised learning, the ANN takes input vector and computes output vector.
- An error signal is generated, if there is a difference between the computed output and the desired output vector. On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.
- This form of learning is called error correction learning.



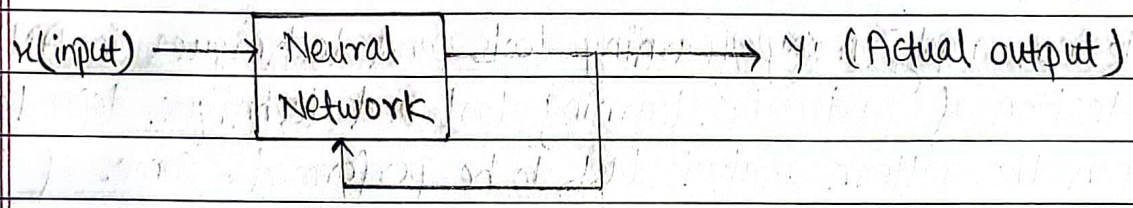
- Supervised machine learning is used for performing tasks like: Regression and Classification.

Learning without Teacher

- In this learning paradigm, we do not provide training set to the neural network to teach it about mapping between input and output.
- There are two types of learning processes under this learning paradigm
  - Unsupervised Learning
  - Reinforcement Learning

Unsupervised Learning

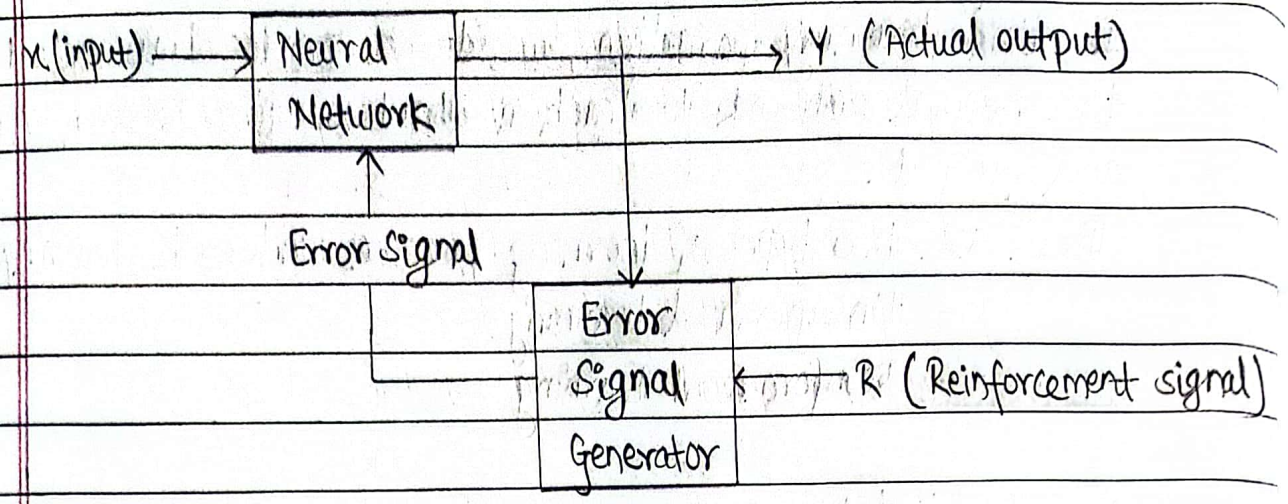
- In unsupervised learning, neural network is provided with dataset without desired output.
- The neural network then attempts to find structure in the data by extracting useful features and analyzing its structure.
- To perform this type of learning, we use competitive learning rule.



- Unsupervised learning algorithms are widely used for tasks like: clustering, dimensionality reduction, association mining, etc.

Reinforcement Learning

- In reinforcement learning, we do not provide the machine with examples of correct input-output pairs, but we do provide a method for the machine to quantify its performance in the form of a reward signal.
- Reinforcement learning methods resemble how humans and animals learn: the machine tries a bunch of different things and is rewarded with performance signal.



- Reinforcement learning algorithms are widely used for training agents interacting with its environment.

Pattern Analysis Tasks

- We can apply many learning tools and techniques in ANN. Selection of particular learning tool and technique depends upon the pattern analysis task to be performed. Some of the major pattern analysis tasks are listed below.
  - Classification
  - Regression
  - Clustering

Classification and Regression

- Classification and Regression are two major categories of prediction problems which are usually dealt with machine learning.
- Both of them are supervised learning approaches. Classification is the process of finding or discovering a model or function which helps to predict class label for a given data.
- Regression is the process of finding a model or function which is used to predict continuous real-valued output for a given data.

- For example, we can build a classification model to categorize bank loan applications as either safe or risky. We can also construct a classification model to identify digits.
- On the other hand, we can build a ~~classification~~ regression model to predict the expenditures of a potential customers on computer equipment given their income and occupation. We can also build a prediction model to predict stock price given historical trading data.

- The classification and regression process works in following two steps : Learning step and Testing step

- Learning step :

  This step is also called training step or training phase. In this step the learning algorithms build a model on the basis of relationship between input and output in the training dataset. This dataset contains input attributes along with output for every input tuple.
- Because the output of each training tuple is provided, this step is also known as supervised learning.

- Testing step :

  In this step, the model is used for prediction. Here the test dataset is used to estimate the accuracy of the model. This dataset contains values of input attributes along with value output attribute.
- However, the model only takes values of input attributes and predicts output of each input tuple.
- Then, efficiency measures of the model is computed by looking at predicted output and actual output of test dataset. The model can be applied to the new data tuples if the accuracy is considered acceptable.

Clustering

- Unlike classification and prediction, which analyze output-labeled data objects, clustering analyzes data objects without consulting a known output.
- Clustering can be used to generate output labels. The objects are clustered or grouped based on the principle of maximizing the intra-class similarity and minimizing the interclass similarity.
- That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters.
- For example, cluster analysis can be performed on customer data to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing.

	Supervised Learning	Unsupervised Learning
1.	SL algorithms are trained using labelled data.	1. UnSL algorithms are trained using unlabelled data.
2.	S.L. Model takes direct feedback to check if it is predicting correct o/p or not.	2. Un.S.L model does not take any feedback.
3.	S.L model predicts the output.	3. UnSL model finds the hidden patterns in data.
4.	In SL, input data is provided to the model along with o/p.	4. In UnSL, only input data is provided to the model.
5.	The goal of S.L. is to train the model so that it can predict the output when it is given new data.	5. The goal of UnSL is to find the hidden patterns and useful insights from the unknown dataset.
6.	SL needs supervision to train the model.	6. UnSL does not need any supervision to train the model.
7.	SL can be categorized in classification and regression problems.	7. UnSL can be classified in Clustering and associations problem.
8.	It can be used for those cases where we know the input as well as corresponding o/ps.	8. It can be used for those cases where we have only input data and no o/p data.
9.	It produces an accurate result.	9. It may give less accurate result compared to SL.
10.	It is not close to true AI as in this, we first train the model for each data and then only it can predict the correct o/p.	10. It is more close to the true AI as it learns similarly as a child learns daily routine things by his experiences.
11.	It includes various algorithms like as Regression, Logistic Regression, Support Vector Machine, Multi-class classification, Decision tree, Bayesian logic, etc.	11. It includes various algorithms such as clustering, KNN and Apriori algorithm.

Date
Page

UNIT 2:

Linear Models for Regression and Classification

Polynomial Curve Fitting.

- Polynomial curve fitting is a form of regression analysis in which the relationship between the independent variables and dependent variables are modeled in the mth degree polynomial.

- Polynomial Regression models are usually fit with the method of least squares.

- If we assume that the relationship is a linear one, then we can use linear equation given as:

$$y = w_0 + w_1 x.$$

- However, if we assume that the relationship is non-linear, we can use polynomial of more than degree one given as below:

$$y = w_0 + w_1 x + w_2 x^2 + \dots + w_m x^m$$

- Here, w_i, $i = 0, 1, \dots m$ are coefficients of polynomial that needs to be determined minimizing mean squared error. Error function for the n data points is given by:

$$E = \frac{1}{2n} \sum_{i=1}^{n} e_i^2$$

$$E = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - w_0 - w_1 x_i - w_2 x_i^2 - \dots - w_m x_i^m\right)^2$$

- Now, coefficients can be determined or updated using gradient descent method as below:

$$w_0 = w_0 - \alpha \frac{\partial E}{\partial w_0}$$

$$= w_0 + \alpha \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i - w_2 x_i^2 - \dots - w_m x_i^m) \right)$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$= w_1 + \alpha \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i - w_2 x_i^2 - \dots - w_m x_i^m) \right)$$

$$w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2}$$

$$= w_2 + \alpha \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i - w_2 x_i^2 - \dots - w_m x_i^m) x_i^2 \right)$$

So on.

Example:

Fit the quadratic curve through the following data. Show one epoch of computation.

x	1	2	3	4
$f(x)$	2	7	14	23

Soln:

General form of quadratic equation is: $y = w_0 + w_1 x + w_2 x^2$.

Let us assume that initial values of parameters are:

$w_0 = w_1 = w_2 = 0$

Iteration 1: $x = 1, \; y = f(x) = 2 \quad \alpha = 0.01$

$$w_0 = w_0 + \alpha (y - w_0 - w_1 x - w_2 x^2)$$
$$= 0 + 0.01 \times 2 = 0.02$$

$$w_1 = w_1 + \alpha (y - w_0 - w_1 x - w_2 x^2) x$$
$$= 0 + 0.01 (2) \times 1 = 0.02$$

$$w_2 = w_2 + \alpha (y - w_0 - w_1 x - w_2 x^2) x^2$$
$$= 0 + 0.01 (2) \times 1^2 = 0.02$$

Iteration 2: $x = 2, \; y = f(x) = 7 \quad \alpha = 0.01$

$$w_0 = w_0 + \alpha (y - w_0 - w_1 x - w_2 x^2)$$
$$= 0.02 + 0.01 (7 - 0.02 - 0.02 \times 2 - 0.02 \times 2^2)$$
$$= 0.02 + 0.01 * 6.86$$
$$= 0.0886$$

$$w_1 = w_1 + \alpha (y - w_0 - w_1 x - w_2 x^2) x$$
$$= 0.02 + 0.01 (7 - 0.02 - 0.02 \times 2 - 0.02 \times 2^2) \times 2$$
$$= 0.02 + 0.01 * 6.86 \times 2$$
$$= 0.1572$$

$$w_2 = w_2 + \alpha (y - w_0 - w_1 x - w_2 x^2) x^2$$
$$= 0.02 + 0.01 (7 - 0.02 - 0.02 \times 2 - 0.02 \times 2^2) \times 2^2$$
$$= 0.02 + 0.01 \times 6.86 \times 4$$
$$= 0.2944$$

Iteration 3: $x = 3$, $y = f(x) = 14$, $\alpha = 0.01$

$$w_0 = w_0 + \alpha(y - w_0 - w_1 x - w_2 x^2)$$
$$= 0.0886 + 0.01(14 - 0.0886 - \overset{0.1572}{\cancel{0.0886}} \times 3 - \cancel{0.88}\ 0.2944 \times 3^2$$
$$= 0.1965$$

$$w_1 = w_1 + \alpha(y - w_0 - w_1 x - w_2 x^2)\, x$$
$$= 0.1572 + 0.01(14 - 0.0886 - 0.1572 \times 3 - 0.2944 \times 3^2)\, 3$$
$$= 0.4809$$

$$w_2 = w_2 + \alpha(y - w_0 - w_1 x - w_2 x^2)\, x^2$$
$$= 0.2944 + 0.01(14 - 0.0886 - 0.1572 \times 3 - 0.2944 \times 3^2) \times 3^2$$
$$= 1.2655$$

Iteration 4: $x = 4$, $y = f(x) = 23$, $\alpha = 0.01$

$$w_0 = w_0 + \alpha(y - w_0 - w_1 x - w_2 x^2)$$
$$= 0.1965 + 0.01(23 - 0.1965 - 0.4809 \times 4 - 1.2655 \times 4^2)$$
$$= 0.2028$$

$$w_1 = w_1 + \alpha(y - w_0 - w_1 x - w_2 x^2)\, x$$
$$= 0.4809 + 0.01(23 - 0.1965 - 0.4809 \times 4 - 1.2655 \times 4^2) \times 4$$
$$= 0.5062$$

$$w_2 = w_2 + \alpha(y - w_0 - w_1 x - w_2 x^2)\, x^2$$
$$= 1.2655 + 0.01(23 - 0.1965 - 0.4809 \times 4 - 1.2655 \times 4^2) \times 4^2$$
$$= 1.3666$$

Logistic Regression.

- Logistic regression is one of the most popular machine learning algorithms for binary classification. This is because it is a simple algorithm that performs very well on a wide range of problems.

- We want to predict a variable $\hat{y} \in \{0, 1\}$, where 0 is called negative class, while 1 is called positive class. Such task is known as binary classification.

- The heart of the logistic regression technique is logistic function and is defined as:

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

- Logistic function transforms the input into the range [0,1]. Smallest negative number results in values close to zero and the larger positive numbers results in values close to one.

- If there are two input variable, logistic regression has two coefficients just like linear regression.

$$y = w_0 + w_1 x_1 + w_2 x_2$$

- Unlike linear regression, the output is transformed into a probability using the logistic function:

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

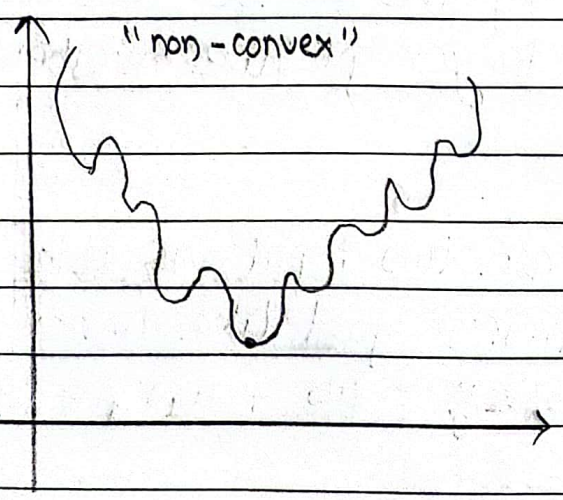
- If the probability is > 0.5 we can take the output as a prediction for the class 1, otherwise the prediction is for the class 0.

- The job of the learning algorithm will be to discover the best values for the coefficients (w_0, w_1, and w_2) based on the training data.

Logistic Regression Cost Function

- Unfortunately, we can't use the cost function MSE (Mean squarred Error) in logistic regression. It is because our prediction function is non-linear (due to sigmoid transform).

- Squarring this prediction as we do in MSE results in a non-convex function with many local minimums. If our cost function has many local minimums, gradient descent may not find the optimal global minimum.



- Given the training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
  we want $\hat{y}^{(i)} \approx y^{(i)}$

- For logistic regression, we use following loss function or error function

$$L(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

- In case $y = 1$, the output (the cost to pay) approaches to 0 as $\hat{y}$ approaches to 1. Conversely, the cost to pay grows to infinity as $\hat{y}$ approaches to 0.

- This is a desirable property: we want a bigger penalty as the algorithm predicts something far away from the actual value. The same intuition applies when $y = 0$.

- Thus, cost function for Logistic regression is given as:
  $L(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$

Logistic Regression GD | Derivatives.

$$\frac{\partial L(y,\hat{y})}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial L(y,\hat{y})}{\partial y} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} = \left\{ -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \right\} \{ \hat{y}(1-\hat{y}) \} = \hat{y} - y$$

$\because$ if $g(x)$ is logistic function $g'(x) = g(x)(1-gx)$

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_0} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_1} = x_1 (\hat{y} - y)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{d\hat{y}} \frac{\partial \hat{y}}{\partial y} \frac{\partial y}{\partial w_2} = x_2 (\hat{y} - y)$$

- Thus, parameters of Logistic Regression are updated as below:

$$w_0 = w_0 - \alpha \frac{\partial L}{\partial w_0}$$

$$w_1 = w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w_2 = w_2 - \alpha \frac{\partial L}{\partial w_2}$$

Date
Page

Example of Logistic Regression.

Fit the logistic regression model through the following data. Show one epoch of training.

x_1	x_2	Class (y)
0.78	0.69	1
0.67	1.00	1
0.00	0.00	0
0.22	0.14	0

Sol:

General form of logistic regression equation is :

$$(z)\ y = w_0 + w_1 x_1 + w_2 x_2$$

Let us assume that initial values of parameters are:

$$w_0 = w_1 = w_2 = 0$$

Iteration 1: $x_1 = 0.78$, $x_2 = 0.69$, $y = 1$, $\alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$\hat{y} = \frac{1}{1+e^{-y(z)}} = \frac{1}{1+e^{-0}} = \frac{1}{1+1} = \frac{1}{2} = 0.5$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0 - 0.1 \times (0.5 - 1) = 0.05$$

$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0 - 0.1(0.5 - 1) \times 0.78 = 0.039$$

$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0 - 0.1(0.5 - 1) \times 0.69 = 0.0345$$

Iteration 2: $x_1 = 0.67$, $x_2 = 1.00$, $y = 1$, $\alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.05 + 0.039 \times 0.67 + 0.0345 \times 1$$
$$= 0.05 + 0.026 + 0.0345$$
$$= 0.1105$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.1105}} = 0.5276$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0.05 - 0.1(0.5276 - 1) = 0.097$$
$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.039 - 0.1(0.5276 - 1)\,0.67 = 0.0706$$
$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0345 - 0.1(0.5276 - 1)\,1 = 0.0817$$

Iteration 3: $x_1 = 0.00$, $x_2 = 0.00$, $y = 0$, $\alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.097 + 0.0706 \times 0 + 0.0817 \times 0$$
$$= 0.097$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.097}} = 0.524$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0.097 - 0.1(0.524 - 0) = 0.0446$$
$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.0706 - 0.1(0.524 - 0)\,0 = 0.0706$$
$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0817 - 0.1(0.524 - 0)\,0 = 0.0817$$

Date
Page

Iteration 4: $x_1 = 0.22$, $x_2 = 0.14$, $y = 0$, $\alpha = 0.1$

$$y = w_0 + w_1 x_1 + w_2 x_2$$
$$= 0.0446 + 0.0706 \times 0.22 + 0.0817 \times 0.14$$
$$= 0.0446 + 0.0155 + 0.0114$$
$$= 0.0715$$

$$\hat{y} = \frac{1}{1+e^{-y}} = \frac{1}{1+e^{-0.0715}} = 0.5178$$

$$w_0 = w_0 - \alpha(\hat{y} - y) = 0.0446 - 0.1(0.5178 - 0) = -0.0072$$
$$w_1 = w_1 - \alpha(\hat{y} - y)x_1 = 0.0706 - 0.1(0.5178 - 0) \times 0.22 = 0.0592$$
$$w_2 = w_2 - \alpha(\hat{y} - y)x_2 = 0.0817 - 0.1(0.5178 - 0) \times 0.14 = 0.0744$$

Linear Basis Function Models.

- Given a supervised learning problem of using inputs p to predict a continuous target y, the simplest model to use would be linear regression.

- However, what if we know that the relationship between the inputs and the target is non-linear, but we are unsure of exactly what form this relationship has?

- One way to overcome this problem is to use linear basis function models. These models assume that the target is a linear combination of a set of $p+1$ basis functions.

$$y_i = w_0 + w_1 \varphi_1(x_1) + w_2 \varphi_2(x_2) + \dots + w_p \varphi_p(x_p)$$

- This is a generalization of linear regression that essentially replaces each input with a function of the input. A linear basis function model that uses the identity function is just linear regression.

- The example of polynomial regression is a particular example of this model in which there is a single input variable x, and the basis functions take the form of powers of x so that $\varphi_j(x) = x^j$

- There are many other possible choices for the basis functions, for example.

- Gaussian basis functions :

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

- Sigmoidal basis functions :

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$$

where,

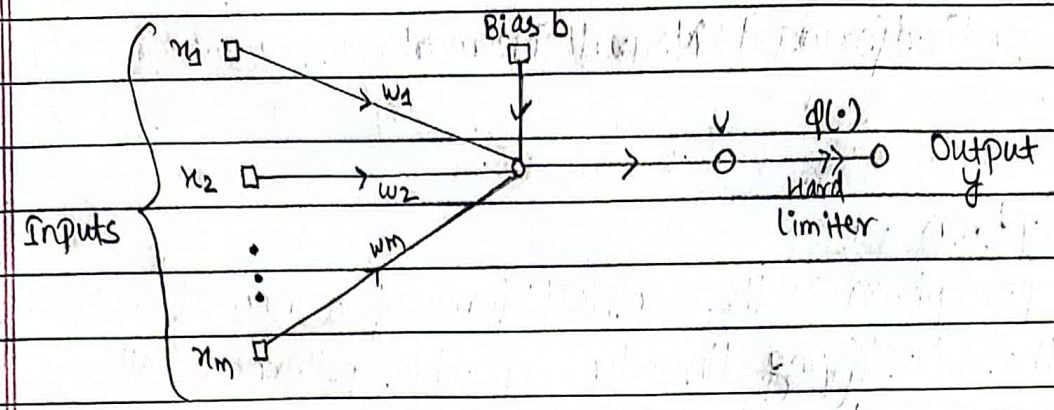
$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

UNIT 3:

Feedforward Neural Networks

Perceptron

- The perceptron is the simplest form of a neural network used for the classifying linearly separable patterns. Patterns that lie on opposite sides of a hyperplane are called linearly separable patterns.
- Basically, perceptron consists of a single neuron with adjustable synaptic weights and bias. The algorithm used to adjust the free parameters of this neural network first appeared in a learning procedure developed by Rosenblatt (1958, 1962)
- Rosenblatt proved that if the patterns (vectors) used to train the perceptron are drawn from two linearly separable classes, then the perceptron algorithm converges and positions the decision surface in the form of a hyperplane between the two classes.
- The perceptron built around a single neuron is limited to performing pattern classification with only two classes. By expanding the output layer of the perceptron to include more than one neuron, we may correspondingly perform classification with more than two classes. However, the classes have to be linearly separable for the perceptron to work properly.
- Rosenblatt's perceptron is built around the McCulloch-Pitts model of a neuron.
- The summing node of the neural model computes a linear combination of the input. The resulting sum is applied to a hard limit activation function.
- The neuron produces an output equal to 1 if the hard limiter input is positive, and -1 if it is negative.

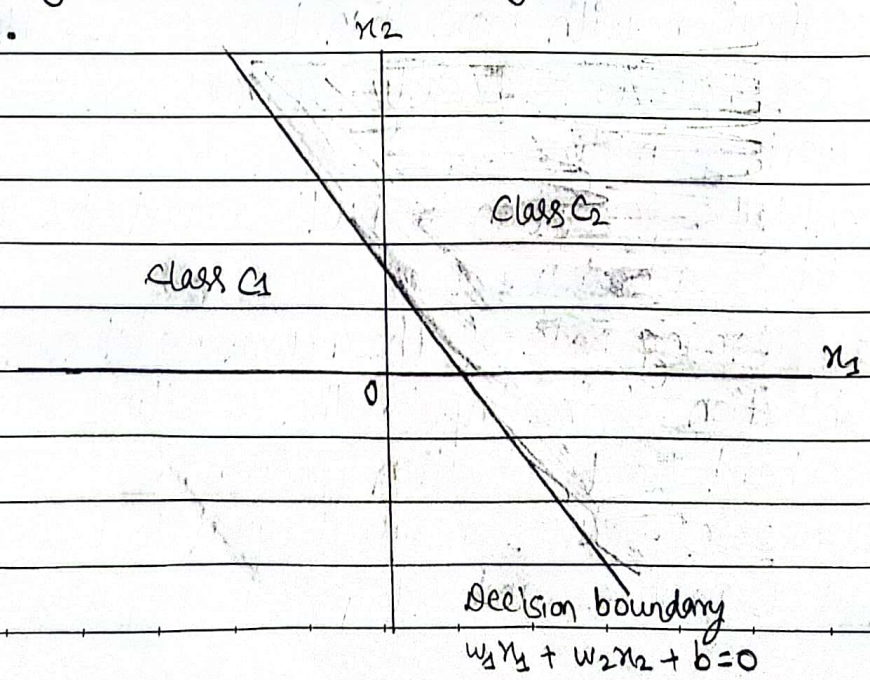


The hard limiter input (or induced local field) of the neuron is.

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

- The goal of the perceptron is to correctly classify the set of externally applied stimuli $x_1, x_2, \ldots, x_m$ into one of two classes, C_1 or C_2. The decision rule for the classification is to assign the point represented by the inputs $x_1, x_2, \ldots, x_m$ to class C_1 if the perceptron output y is $+1$ and to class C_2 if it is -1.

- In the simplest form of the perceptron, there are two decision regions separated by a hyperplane, which is defined by.



- Perceptron can be trained to show behavior of AND function and OR function because both are linearly separable. But, perceptron can not be trained to learn behavior of XOR function because it is not linearly separable.

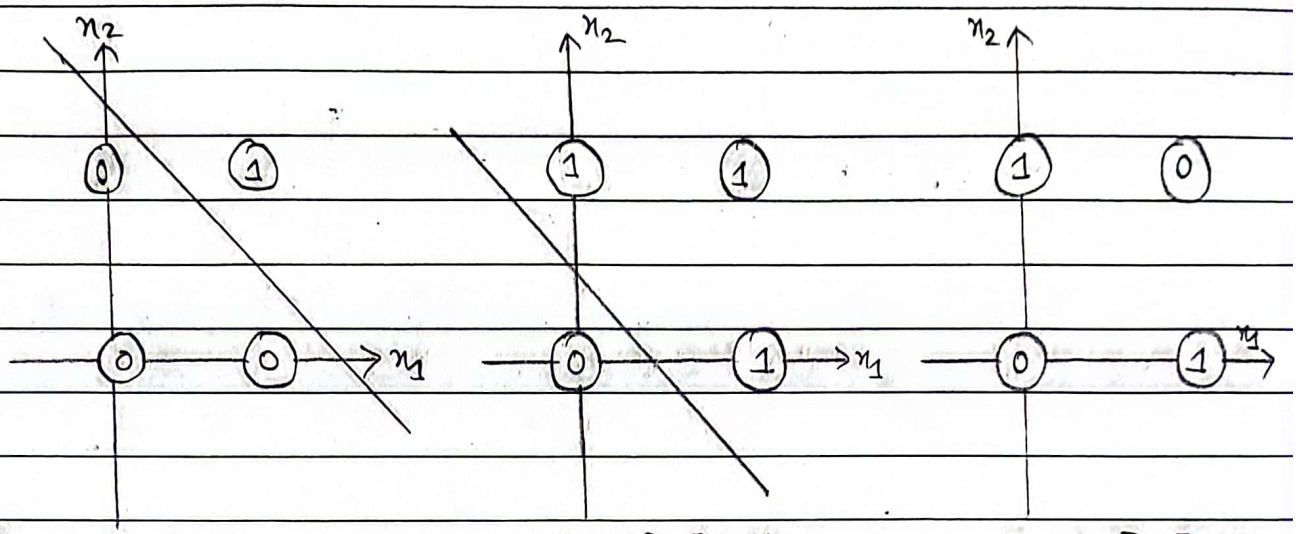


Fig: AND Function Fig: OR Function Fig: XOR Function

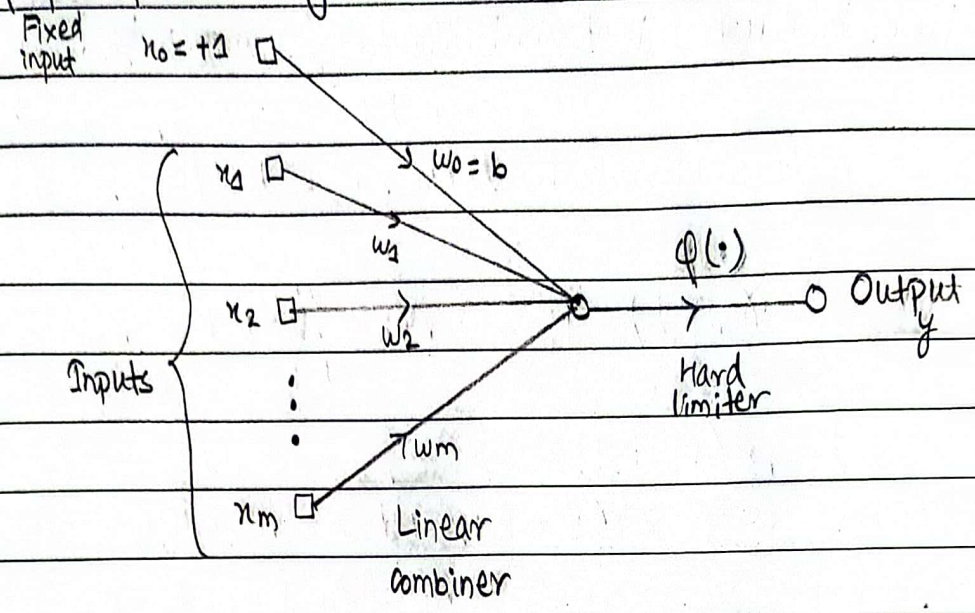
Perceptron Convergence Theorem

- The perceptron convergence theorem states that "for any data set which is linearly separable, the Perceptron learning rule is guaranteed to find a solution in a finite number of steps, provided that learning rate is small."

- Perceptron learning rule is guaranteed to converge to a weight vector that correctly classifies the examples provided the training examples are linearly separable.

Perceptron Learning Rule

- We can derive error correction rule for perceptron. Model of perceptron is given below;



- For this model, we can write

$$V = \sum_{i=0}^{m} w_i x_i$$

- Let $x = [x_0, x_1, x_2, \ldots, x_m]$ and $w = [w_0, w_1, \ldots, w_m]$
- Now, above equation can be written as:

$$V = w^T x$$

- Let us consider for n^{th} step input vector and weight vector is:

$$x = [x_0(n), x_1(n), \ldots x_m(n)] \quad \text{and}$$
$$w = [w_0(n), w_1(n), \ldots w_m(n)]$$

- Now, for n^{th} step, above equation becomes

$$V = w(n)^T x(n)$$

- Suppose C_1 and C_2 are two classes for which perceptron needs to be trained. So, training set must include sufficient examples from both classes.

- Let H_1 and H_2 are set of training examples such that $H_1 \in C_1$ and $H_2 \in C_2$. Thus, Training Set $(H) = H_1 \cup H_2$.
- We know that perceptron puts data point in class C_1 if output is positive and in class C_2 if output is negative.
- This means, data point lies in class C_1 if $v > 0$ and in class C_2 if $v \leq 0$. Thus,
  - For each $x \in H_1$, $w^T x > 0$ and
  - For each $x \in H_2$, $w^T x \leq 0$

- Now, we can write weight update rules for n^{th} step as below:

Case I : Correct Classification

if $w^T(n) x(n) > 0$ and $x(n) \in C_1$ then $w(n+1) = w(n)$

if $w^T(n) x(n) \leq 0$ and $x(n) \in C_2$ then $w(n+1) = w(n)$

Case II : Missclassification

if $w^T(n) x(n) > 0$ and $x(n) \in C_2$ then $w(n+1) = w(n) - \alpha x(n)$

if $w^T(n) x(n) \leq 0$ and $x(n) \in C_1$ then $w(n+1) = w(n) + \alpha x(n)$.

- The main theme of above weight change rule is:
  - Do not change weights in case of correct classification.
  - Increase weights when perceptron output is smaller than actual target.
  - Decrease weights when perceptron output is larger than actual target.

- Thus, we can ~~change~~ represent weight change rule using single eq^n as below:

$$\boxed{w(n+1) = w(n) + \alpha\, x(n) \left(t(n) - y(n) \right)}$$

where, $y(n)$ is actual output, and
$t(n)$ is target output

Perceptron Learning Algorithm

1. Initialize all weights and bias to zero.
2. For each training vector s and target t perform steps 3 to 6.
3. set $x_i = s_i$ for $i = 1$ to n.
4. Compute output using Hard limiter activation function as below

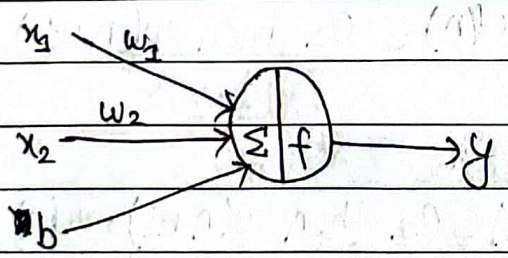
$$y_{in} = b + \sum_{i=1}^{n} w_i x_i \qquad y = f(y_{in})$$

5. Adapt weights as : $w_i = w_i + \alpha(t-y)x_i$ for $i = 1$ to n.
6. Adapt bias as : $b = b + \alpha(t-y)$
7. Test for stopping Criteria

Perceptron Training

Example:

Train the following perceptron by using given training set



x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Sol^n: $\alpha = 1$

Epoch #1

$$v = \sum_{i=1}^{n} w_i x_i + b$$

Input	w_1 (old)	w_2 (old)	b (old)	v	y	w_1(new)	w_2(new)	b(new)
(1,1,1)	0	0	0	0	0	0+1*1*1=1	0+1*1*1=1	0+1*1=1
(1,-1,-1)	1	1	1	1	1	1+1*-2*1=1	1+1*-2*-1=3	1+1*-2=-1
(-1,1,-1)	-1	3	-1	3	1	-1*1*-2*-1=1	3+1*-2*1=1	-1+1*-2=-3
(-1,-1,-1)	1	1	-3	-5	-1	1+1*0*-1=1	1+1*0*-1=1	-3+1*0=-3

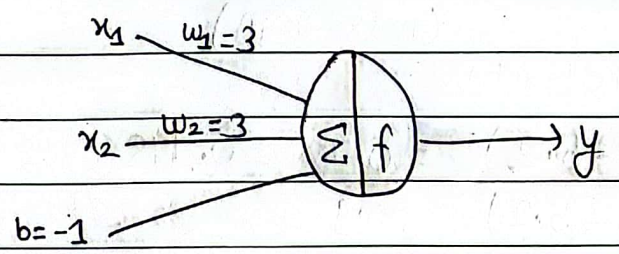
$$\varphi(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \end{cases}$$

Assumed hard Limiter Activation function.

Epoch #2

Input	w_1 (old)	w_2 (old)	b (old)	v	y	w_1(new)	w_2(new)	b(new)
(1,1,1)	1	1	-3	-1	-1	1+1*2*1=3	1+1*2*1=3	-3+1*2=-1
(1,-1,-1)	3	3	-1	-1	-1	3+1*0*1=3	3+1*0*(-1)=3	-1+1*0=-1
(-1,1,-1)	3	3	-1	-1	-1	2+1*0*-1=3	3+1*0*1=3	-1+1*0=-1
(-1,-1,-1)	3	3	-1	-7	-1	3+1*0*-1=3	3+1*0*-1=3	-1+1*0=-1

Thus, Final Neuron is



Example 2:

Train perceptron using given training set and predict class for the input (6, 82) and (5.3, 52).

Height (x_1)	Weight (x_2)	Class (t)
5.9	75	Male
5.8	86	Male
5.2	50	Female
5.4	55	Female
6.1	85	Male
5.5	62	Female

Sol^n:

Normalize the input and encode target

Use min-max scaling, $y = \frac{(x - \min)}{(\max - \min)}$

Here, Normalizing the input x_1 (i.e. height)

Saling of 5.9: $\frac{(5.9 - 5.2)}{(6.1 - 5.2)} = \frac{0.7}{0.9} = 0.778$

Scaling of 5.8: $\frac{(5.8 - 5.2)}{(6.1 - 5.2)} = \frac{0.6}{0.9} = 0.667$

Scaling of 5.2: $\frac{(5.2 - 5.2)}{(6.1 - 5.2)} = \frac{0}{0.9} = 0.000$

Scaling of 5.4: $\frac{(5.4 - 5.2)}{(6.1 - 5.2)} = \frac{0.2}{0.9} = 0.222$

Scaling of 6.1: $\frac{(6.1 - 5.2)}{(6.1 - 5.2)} = \frac{0.9}{0.9} = 1.000$

Scaling of 5.5: $\frac{(5.5 - 5.2)}{(6.1 - 5.2)} = \frac{0.3}{0.9} = 0.333$

Similarly, normalize other values (i.e. weights)

Weights:

$75 : \frac{(75-50)}{(86-50)} = \frac{25}{36} = 0.694$

$86 : \frac{(86-50)}{(86-50)} = \frac{36}{36} = 1.000$

$50 : \frac{(50-50)}{(86-50)} = \frac{0}{36} = 0.000$

$55 : \frac{(55-50)}{(86-50)} = \frac{5}{36} = 0.139$

$85 : \frac{(85-50)}{(86-50)} = \frac{35}{36} = 0.972$

$62 : \frac{(62-50)}{(86-50)} = \frac{12}{36} = 0.333$

Then,

Heights (x_1)	Weight (x_2)	Class (t)		Height (x_1)	Weight (x_2)	Class (t)
5.9	75	Male		0.778	0.694	1
5.8	86	Male		0.667	1.000	1
5.2	50	Female	$\Rightarrow$	0.000	0.000	-1
5.4	55	Female		0.222	0.139	-1
6.1	85	Male		1.000	0.972	1
5.5	62	Female		0.333	0.333	-1

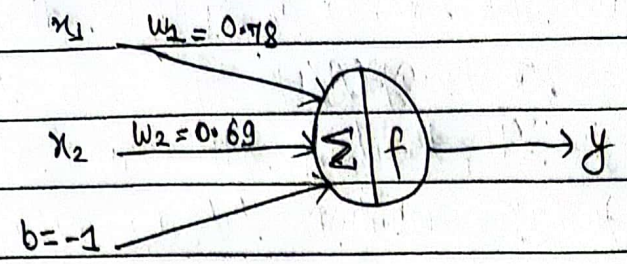
$\alpha = 1$

Epoch #1

$y = w_1 x_1 + w_2 x_2 + b$ $\quad w_i = w_i + \alpha (t-y) x_i$ $\quad b = b + \alpha(t-y)$

Input	w_1 (old)	w_2 (old)	b (old)	y	y	w_1(new)	w_2(new)	b(new)
0.78, 0.69, 1	0	0	0	0	0	0+1*1*0.78=0.78	0+1*1*0.69=0.69	0+1*1=1
0.67, 1, 1	0.78	0.69	1	2.21	1	0.78	0.69	1
0, 0, -1	0.78	0.69	1	1	1	0.78+1*(-2)*0=0.78	0.69+1*(-2)*0=0.69	1+1*(-2)=-1
0.22, 0.14, -1	0.78	0.69	-1	-0.73	-1	0.78	0.69	-1
1, 0.97, 1	0.78	0.69	-1	0.45	1	0.78	0.69	-1
0.33, 0.33, -1	0.78	0.69	-1	-0.51	-1	0.78	0.69	-1

Thus, Final perceptron is as below:



For (6, 82)

Normalized value of $6 = \frac{(6-5.2)}{(6.1-5.2)} = \frac{0.8}{0.9} = 0.89$

Normalized value of $82 = \frac{(82-50)}{(86-50)} = \frac{32}{36} = 0.89$

$$v = w_1 x_1 + w_2 x_2 + b$$
$$= 0.78 * 0.89 + 0.69 * 0.89 - 1$$
$$= 0.31$$

Thus, $y = 1$.

Hence, Predicted class for (6, 82) is Male.

Now,

For (5.3, 52)

Normalized value of $5.3 = \frac{(5.3-5.2)}{(6.1-5.2)} = \frac{0.1}{0.9} = 0.11$

Normalized value of $52 = \frac{(52-50)}{(86-50)} = \frac{2}{36} = 0.056$

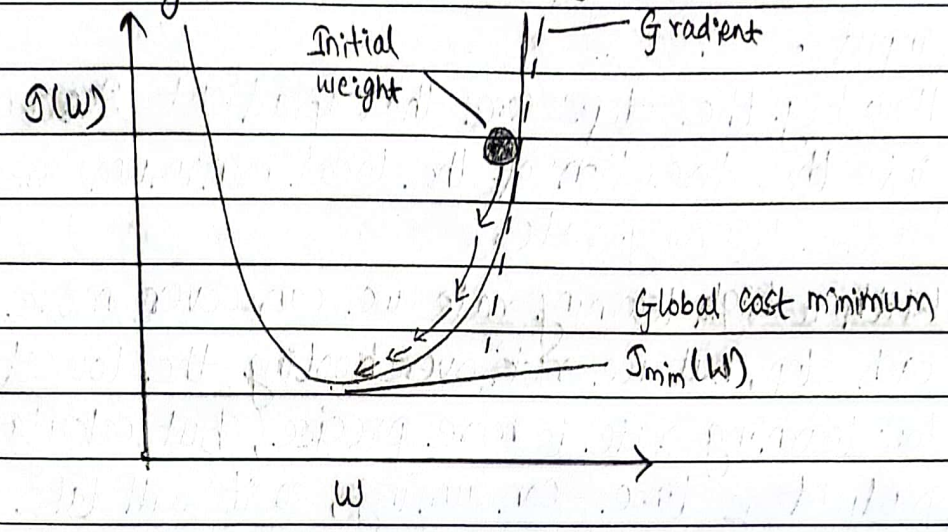
$$v = w_1 x_1 + w_2 x_2 + b$$
$$= 0.78 * 0.11 + 0.69 * 0.05 - 1$$
$$= -0.8797$$

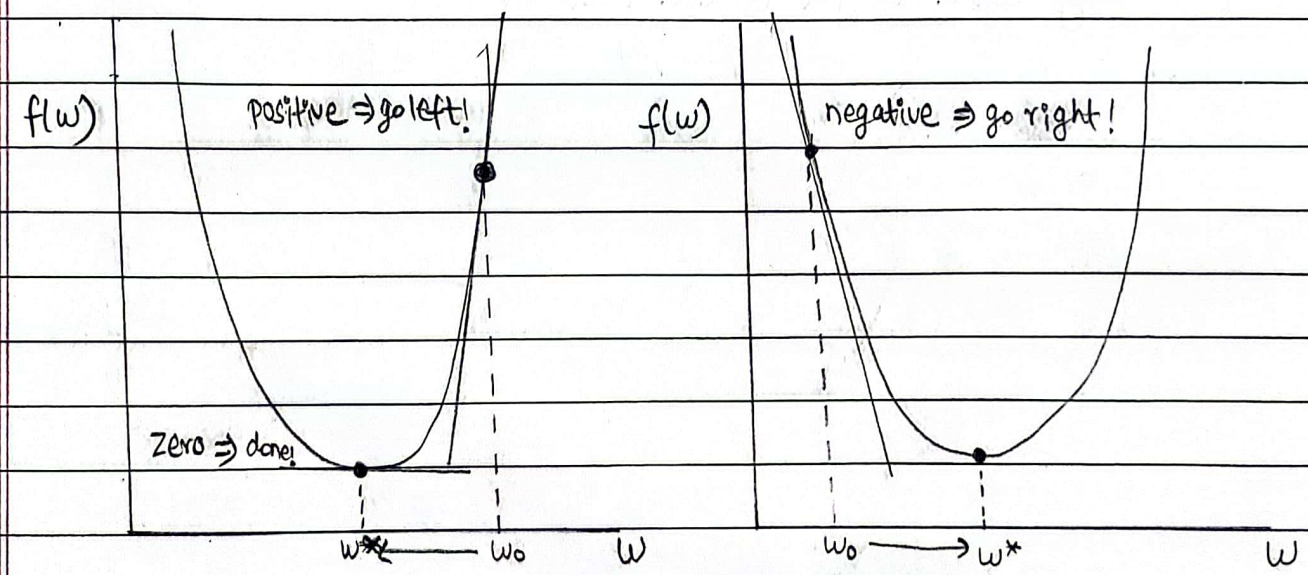
Thus, $y = -1$

Hence, Predicted class for (5.3, 52) is Female.

Gradient Descent

- Gradient Descent is an optimization algorithm used to minimize some convex function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.
- In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to coefficients in Logistic Regression and weights in neural networks.



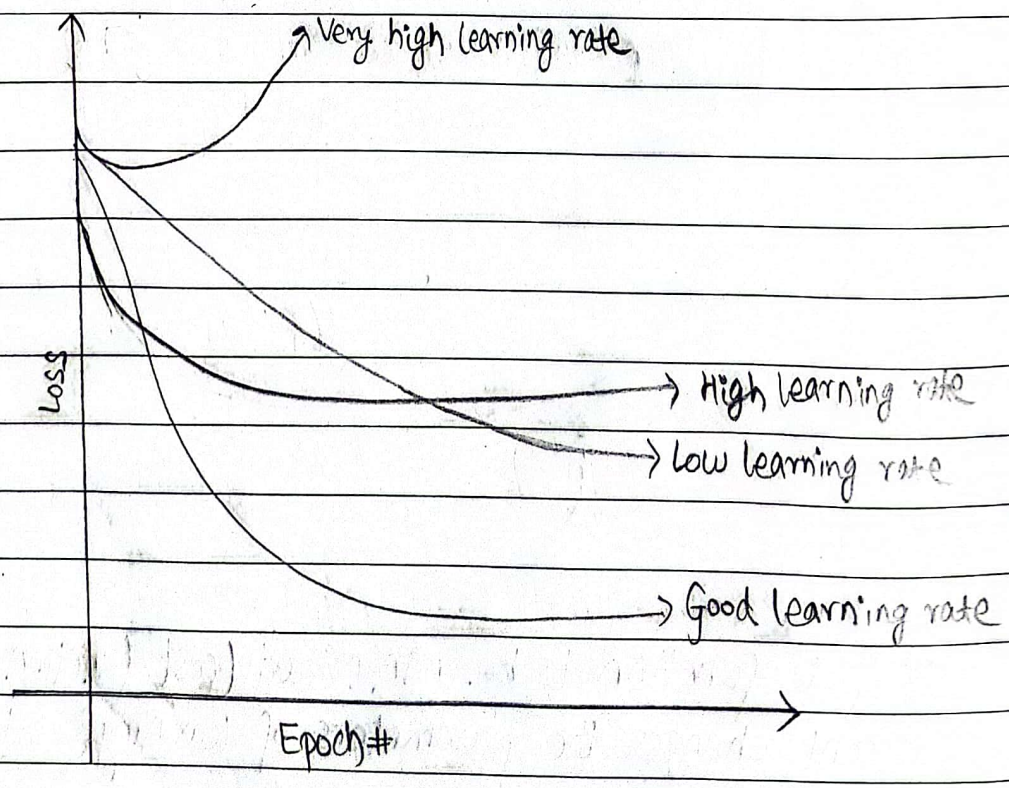


- If f is function to be minimized (cost function), Gradient descent changes the parameters of learning model iteratively as below:

$$w = w - \alpha \frac{df}{dw}$$

where, α is learning rate and w is parameter to be optimized.

- GD simply measures the change in all weights with regard to the change in error. We can also think of a gradient as the slope of a function.
- The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning. Said it more mathematically, a gradient is a partial derivative with respect to its inputs.
- How big the steps are that Gradient Descent takes into the direction of the local minimum are determined by the learning rate.
- With a high learning rate we can cover more ground each step, but we risk overshooting the lowest point. A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom.



Date
Page

Batch Perceptron Algorithm

- When training set is large, taking each training example one by one and updating perceptron weights is time consuming tasks.
- In such case batch perceptron algorithm is suitable. The main concept behind batch perceptron algorithm is:
  "Process training examples in batches. Size of batch should be defined on the basis of size of training set. Commonly used batch size are 16, 32, 64, 128, etc."
- Then, update weights for $(n+1)^{th}$ step by taking sum of all misclassifications in the batch of training example.
- For the training examples that are incorrectly classified by the algorithm:

$$w^T x t > 0$$

- Therefore, to minimize cost function for perceptron, we can write:

$$J(w) = \sum_{x_i \in m} - w^T x_i t_i$$

where, m is the set of classified training examples.

- Differentiate above cost function w.r.t w to get gradient vector.

$$\nabla J(w) = \sum_{x_i \in m} - x_i t_i$$

- In the steepest descent method, the adjustment to the weight vector 'w' at each time step of the algorithm is applied in a direction opposite to the gradient vector.
- Thus, weight update rule for batch perceptron algorithm can be written as:

$$w(n+1) = w(n) - \alpha \nabla J(w) = w(n) + \alpha \sum_{x_i \in m} x_i t_i .$$

Multi-Layer Feedforward NN

- A multilayer feed-forward network consists of an input layer, one or more hidden layers, and an output layer. Computations take place in the hidden and output layers only.
- The input signal propagates through the network in a forward direction layer-by-layer. Such neural networks are called multilayer perceptrons (MLPs).
- They have been successfully applied to many difficult and diverse problems.
- Multilayer perceptrons are typically trained using so-called error backpropagation algorithm. This is a supervised error-correction learning algorithm.

Properties of MLPs

- Each neuron has a smooth (differentiable everywhere) nonlinear activation function. This is usually a sigmoidal nonlinearity defined by the logistic function.

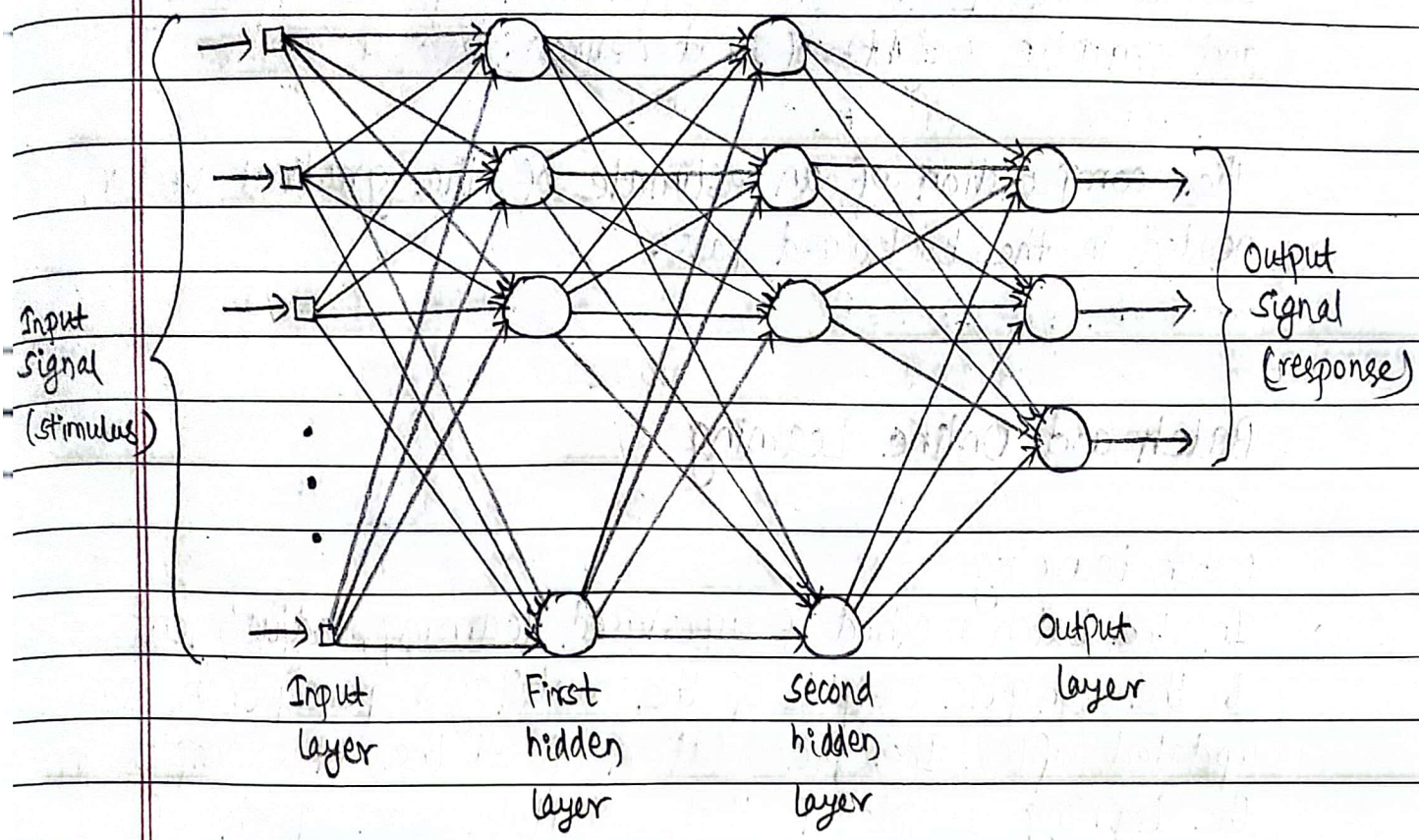
$$y_j = \frac{1}{1 + \exp(-v_j)},$$

where, v_j is weighted sum of inputs

Note: Nonlinearities are important: otherwise the network could be reduced to a linear single-layer perceptron.

- The network contains hidden layer(s), enabling learning complicated tasks and mappings.
- The network has a high connectivity.

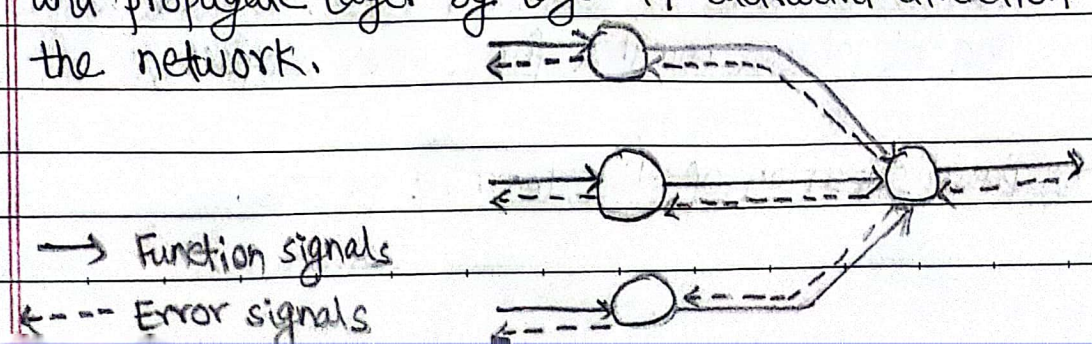
An architectural graph of a fully connected multilayer perceptron with two hidden layers and an output layer is given below.



- Two kinds of signals appear in the MLP networks: Function signal and Error signal.

- Function signals: Function signals are the input signals that propagates in forward direction through the network and produces output signals in the last phase.

- Error Signals: These are the signals originate at output neurons, and propagate layer by layer in backward direction through the network.



- Each hidden or output neuron performs two computations:

1. The computation of the function signal appearing at its output. This is a nonlinear function of the input signal and synaptic weights of that neuron.

2. The computation of an estimate of the gradient vector, needed in the backward pass.

Batch and Online Learning

Batch Learning

- In the batch method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are updated after the presentation of all the N examples in the training sample T that constitute one epoch of training.
- In other words, the cost function for batch learning is defined by the average error energy and adjustments to the synaptic weights of the multilayer perceptron are made on an epoch-by-epoch basis.

- With the method of gradient descent used to perform the training, the advantages of batch learning include the following:
  - Accurage estimation of the gradient vector, thereby guaranteeing, convergence of the method of steepest descent to a local minima for non-convex surfaces and at global minima for convex surface;
  - parallelization of the learning process.

- For large datasets, batch learning is computationally intractable because it is not possible to store entire dataset into memory at once.
- Since it updates weights less frequently, it is computationally faster.

Online Learning

- In the on-line method of supervised learning, adjustments to the synaptic weights of the multilayer perceptron are performed on an example-by-example basis. The cost function to be minimized is therefore the total instantaneous error energy for one training example.

- Due to noise gradient descent calculated for each of the training examples, online learning jumps here and there which makes it difficult to converge in global minima. Due to this behavior of online learning, it is also called stochastic method.

- Besides this, parallelization of online learning method is not possible.

- Since it updates weights example by example basis, it is computationally slower than batch learning. But it is computationally tractable for the large training set because we do not need to store large dataset into memory at once.

Trade-off Between Batch and Online Learning

- If we look above discussion, we can conclude that Batch learning is better approach. But, it is computationally intractable for large datasets.

- On the other hand, online learning is computationally tractable for large datasets but it is difficult to converge in global minim.

- Thus, the better approach is to use mini-batch learning approach, where entire dataset is divided into batches of size 16 or 32 or 64 or 128 etc., on the basis of need and synaptic weights of the multilayer perceptron are updated after the presentation of all the N examples in a mini-batch.

Backpropagation Algorithm

- A popular method for the training of multilayer perceptrons is the backpropagation algorithm. The training proceeds in two phases:

- Forward Phase:

In this phase, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.

- Backward Phase:

In this phase, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated in backward direction through the network, again layer by layer. In this second phase, successive adjustments are made to the synaptic weights of the network.

- The error signal at the output of neuron j at iteration n is given by

$$e_j(n) = d_j(n) - y_j(n)$$

where, $d_j(n)$ is desired output and
$y_j(n)$ is output produced by MLP.

- The total instantaneous error energy $E(n)$ for all the neurons in the output layer is therefore

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

where,
C is the set of neurons in output layer

- Let N be the total number of training vectors (examples). Then the average squarred error is:

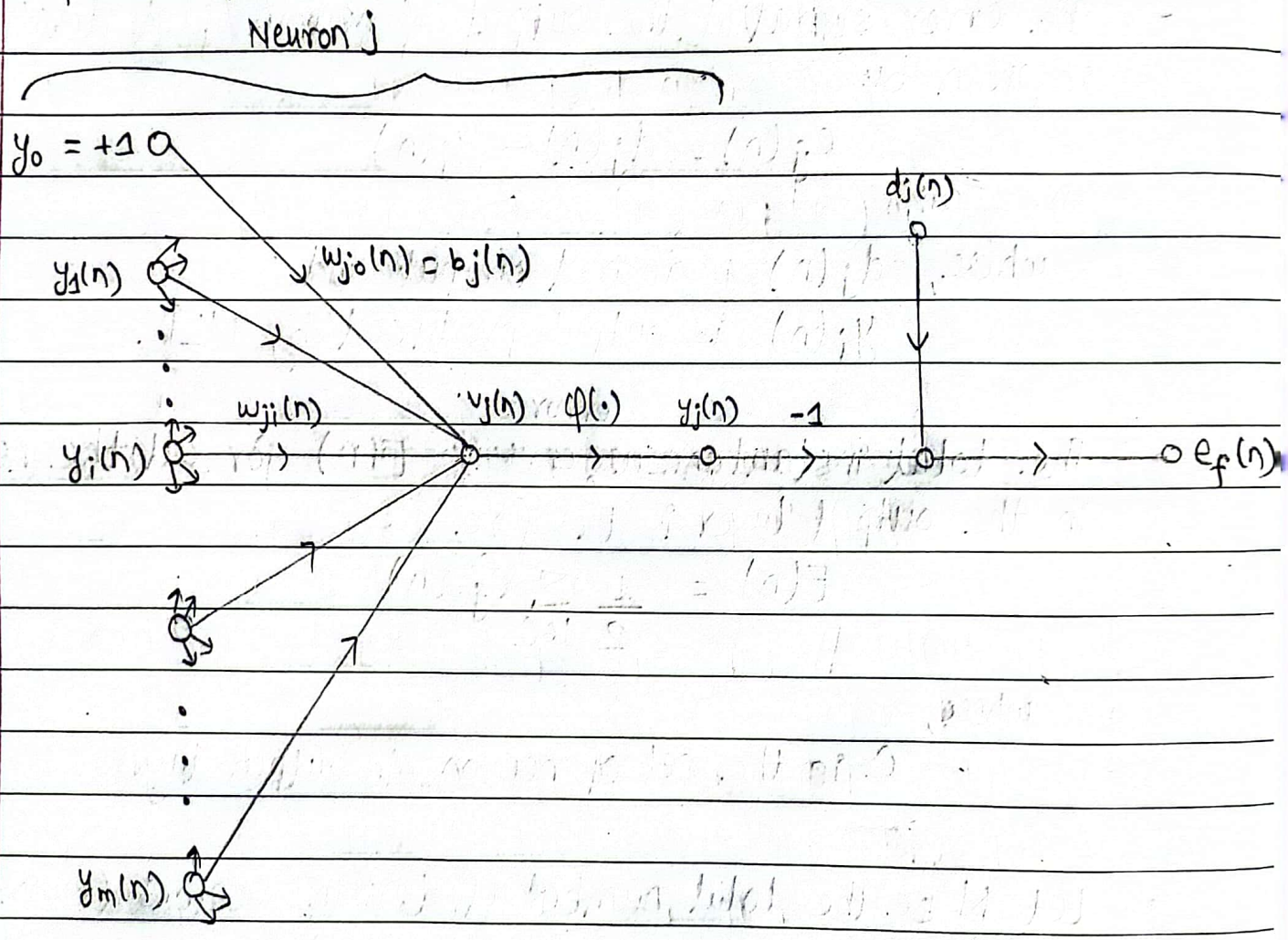
$$E_{avg} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

- Consider the neuron j as given. The local field $v_j(n)$ and output $y_j(n)$ of neuron j is given by:

$$v_j = \sum_{i=0}^{m} w_{ji}(n)\, y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

where y_i is output of neuron i and w_{ji} is weight of link from neuron i to j.



- The correction $\Delta w_{ji}(n)$ made to the weight is proportional to the partial derivative $\partial E(n) / \partial w_{ji}(n)$ of instantaneous error.
- Using the chain rule of calculus, this gradient can be expressed as follows:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \quad \text{— (1)}$$

- We can get following partial derivatives

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \qquad \frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n)) \qquad \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

- Putting all partial derivatives in equation 1, we get

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\, \varphi'_j(v_j(n))\, y_i(n) \quad \text{— (2)}$$

- The correction applied to the weight is defined by

$$\Delta w_{ji} = -\alpha \frac{\partial E(n)}{\partial w_{ji}(n)} \quad \text{— (3)}$$

- Using equation (2), equation (3) can be written as

$$\Delta w_{ji} = \alpha\, e_j(n)\, \varphi'_j(v_j(n))\, y_i(n) \quad \text{— (4)}$$

- This can be written as

$$\Delta w_{ji} = \alpha\, \delta_j(n)\, y_i(n)$$

where,

$$\delta_j(n) = e_j(n)\, \varphi'_j(v_j(n)) \quad \text{— (5)}$$

- The error term in equation (4) and (5) depends upon location of neuron in the MLP. There can be two possible scenarios.

Case I: Neuron j is output layer neuron

- The desired response $d_j(n)$ for the neuron j is directly available. Computation of the error $e_j(n)$ is straightforward in this case. We can use equation 4 or 5 to calculate weight update term.

Case II: Neuron j is hidden layer neuron

- There is no desired response available for neuron j. The error signal for a hidden neuron must be determined recursively in terms of the error signals of all neurons connected to it as below:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \quad \text{--- (6)}$$

Algorithm

1. Initialize all weights and biases in network
2. While terminating condition is not satisfied
3. for each training tuple 'X' in $\mathcal{D}$
4. for each input layer unit j: $y_j = x_j$; // output of an input unit is its actual input value
5. for each hidden layer or output layer unit j
   - compute the net input of unit j with respect to the previous layer, $\boxed{v_j = \sum_i w_{ji} y_i}$
   - compute the output of each unit j, $\boxed{y_j = \frac{1}{1+e^{-v_j}}}$
6. for each unit j in the output layer, $\boxed{\delta_j = e_j \varphi_j'(v_j) = \varphi_j'(v_j)(d_j - y_j)}$
7. for each unit j in the hidden layers, from the last to the first hidden layer. $\boxed{\delta_j = \varphi_j'(v_j) \sum_k \delta_k w_{kj}}$
8. for each weight w_{ji} in network
   $\Delta w_{ji} = \alpha \delta_j y_i$
   $w_{ji} = w_{ji} + \Delta w_{ji}$ // Weight Update

- Backpropagation algorithm needs to compute $\varphi_j'(v_j)$ to compute δ_j. This depends upon choice of activation function.

Case I: Logistic Activation Function

$$\varphi(x) = y = \frac{1}{1+e^{-x}} \Rightarrow \varphi'(x) = y(1-y)$$

If we consider general sigmoid function, then

$$\varphi(x) = y = \frac{1}{1+e^{-ax}} \Rightarrow \varphi'(x) = ay(1-y)$$

Note: Compute Derivatives of Logistic activation function

Case II: Hyperbolic Tangent Function

$$\varphi(x) = y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \Rightarrow \varphi'(x) = (1+y)(1-y)$$

If we consider general Tanh function, then

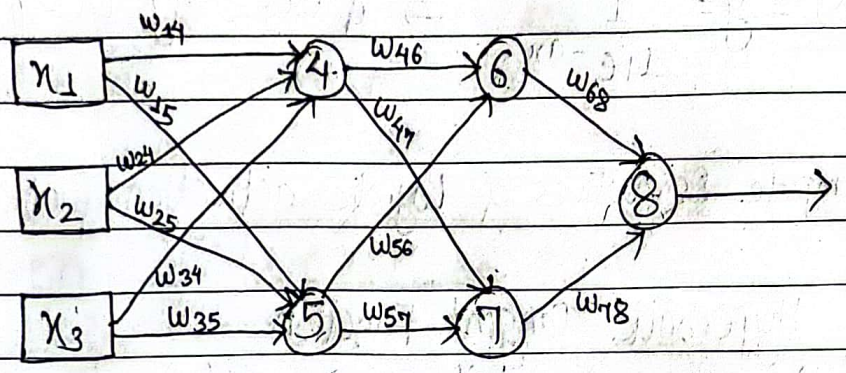
$$\varphi(x) = y = a\tanh(bx) \Rightarrow \varphi'(x) = \frac{b}{a}(a+y)(a-y)$$

Note: Compute Derivatives of Hyperbolic Function.

Example:

Consider a MLP given below. Let the learning rate be 1. The initial weights of the network are given in the table below. Assume that first training tuple is (1,0,1) and its target output is 1. Calculate weight updates by using back-propagation algorithm.

Assume $\varphi(x) = \dfrac{1}{1+e^{-x}}$



w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{47}	w_{56}	w_{57}	w_{68}	w_{78}
0.6	0.4	0.2	-0.3	0.7	-0.6	0.4	0.7	0.1	0.8	0.2	0.5

Sol^n:

Forward Pass:

$V_4 = 1*0.6 + 0*0.2 + 1*0.7 = 1.3$;	$y_4 = \dfrac{1}{1+e^{-1.3}} = 0.786$
$V_5 = 1*0.4 + 0*(-0.3) + 1*(-0.6) = -0.2$;	$y_5 = \dfrac{1}{1+e^{0.2}} = 0.45$
$V_6 = 0.786*0.4 + 0.45*0.1 = 0.36$;	$y_6 = \dfrac{1}{1+e^{-0.36}} = 0.59$
$V_7 = 0.786*0.7 + 0.45*0.8 = 0.91$	$y_7 = \dfrac{1}{1+e^{-0.91}} = 0.71$
$V_8 = 0.59*0.2 + 0.71*0.5 = 0.47$	$y_8 = \dfrac{1}{1+e^{-0.47}} = 0.61$

Backward Pass:

$\delta_8 = y_8(1-y_8)(d_8-y_8) = 0.61*(1-0.61)*(1-0.61) = 0.093$

$\delta_7 = y_7(1-y_7)\,\delta_8 w_{87} = 0.71*(1-0.71)*0.093*0.5 = 0.0096$

$\delta_6 = y_6(1-y_6)\,\delta_8 w_{86} = 0.59*(1-0.59)*0.093*0.2 = 0.0045$

$\delta_5 = y_5(1-y_5)(\delta_7 w_{75} + \delta_6 w_{65})$

$= 0.45*(1-0.45)*(0.0096*0.8 + 0.0045*0.1) = 0.002$

$\delta_4 = y_4(1-y_4)(\delta_7 w_{74} + \delta_6 w_{64})$

$= 0.786*(1-0.786)*(0.0096*0.7 + 0.0045*0.4) = 0.0014$

Update Weights

$w_{14} = w_{14} + 1*\delta_4*y_1 = 0.6 + 1*0.0014*1 = 0.6014$

$w_{15} = w_{15} + 1*\delta_5*y_1 = 0.4 + 1*0.002*1 = 0.402$

$w_{24} = w_{24} + 1*\delta_4*y_2 = 0.2 + 1*0.0014*0 = 0.2$

$w_{25} = w_{25} + 1*\delta_5*y_2 = -0.3 + 1*0.002*0 = -0.3$

$w_{34} = w_{34} + 1*\delta_4*y_3 = 0.7 + 1*0.0014*1 = 0.7014$

$w_{35} = w_{35} + 1*\delta_5*y_3 = -0.6 + 1*0.002*1 = -0.598$

$w_{46} = w_{46} + 1*\delta_6*y_4 = 0.4 + 1*0.0045*0.786 = 0.4035$

$w_{47} = w_{47} + 1*\delta_7*y_4 = 0.7 + 1*0.0096*0.786 = 0.7075$

$w_{56} = w_{56} + 1*\delta_6*y_5 = 0.1 + 1*0.0045*0.45 = 0.1020$

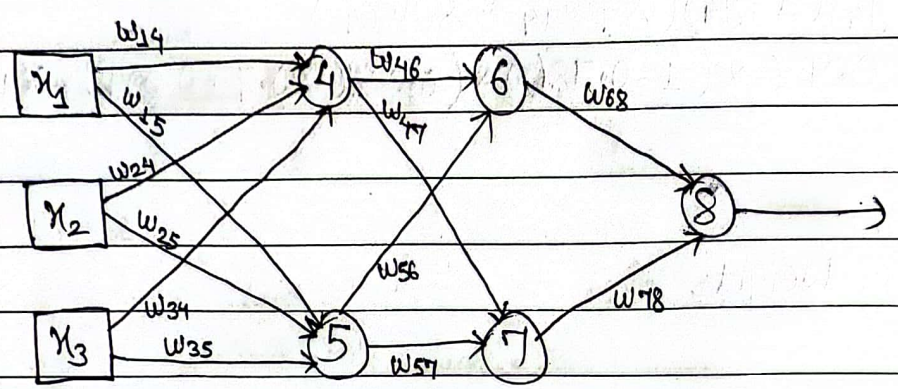
$w_{57} = w_{57} + 1*\delta_7*y_5 = 0.8 + 1*0.0096*0.45 = 0.8043$

$w_{68} = w_{68} + 1*\delta_8*y_6 = 0.2 + 1*0.093*0.59 = 0.2549$

$w_{78} = w_{78} + 1*\delta_8*y_7 = 0.5 + 1*0.093*0.71 = 0.5660$

HW

Consider a MLP given below. Let the learning rate be 1. The initial weights of the network are given in the table below. Assume that first training tuple is (1, 0, 1) and its target output is 1. Calculate weight updates by using back-propagation algorithm. Assume $\phi(x) = y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

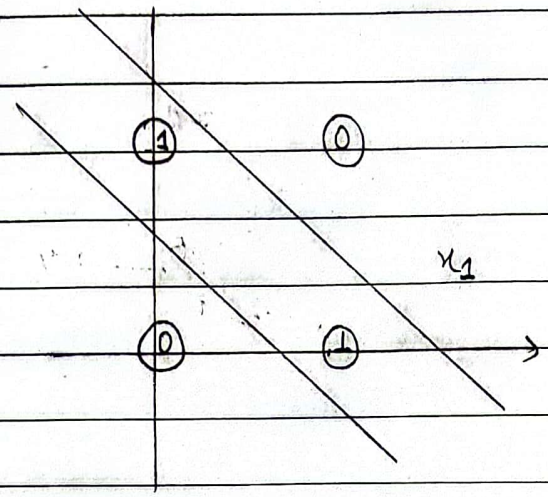


w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{47}	w_{56}	w_{57}	w_{68}	w_{78}
0.6	0.4	0.2	-0.3	0.7	-0.6	0.4	0.7	0.1	0.8	0.2	0.5

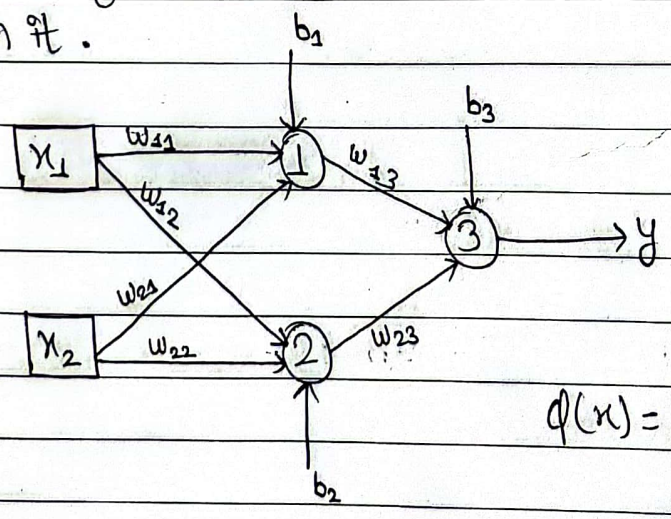
Soln:

XOR Problem

- We know that single layer perceptron can be trained to classify patterns that is linearly separable.
- However, XOR function is non-linearly separable as shown in the figure given below. Thus, we can not train single layer perceptron for XOR function. This problem is called XOR problem.



- We need to use multilayer perceptron that contains one hidden layer with two neurons to achieve XOR functionality from it.



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

- Determine the weights of links for above MLP such that MLP functions as XOR. Assume that threshold activation function (as shown above) is used in each of the neuron.

- We can have following realization of AND, OR, and NOR gates using perceptron.

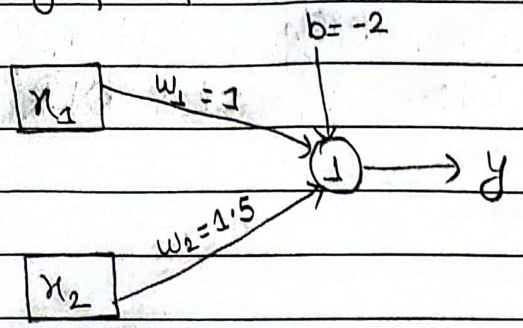


Fig: AND Function

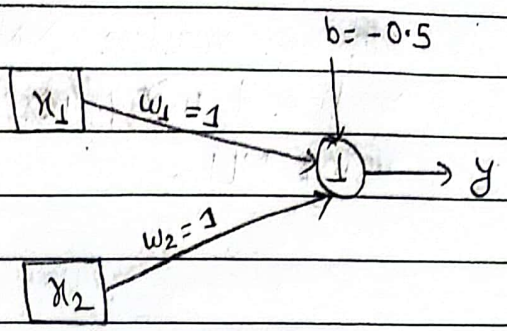


Fig : OR Function

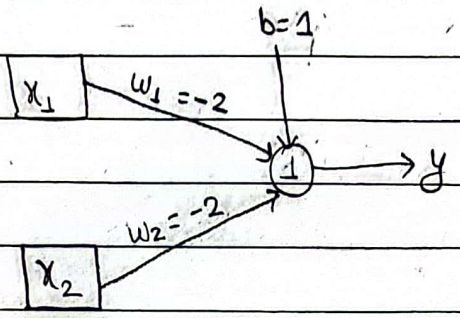
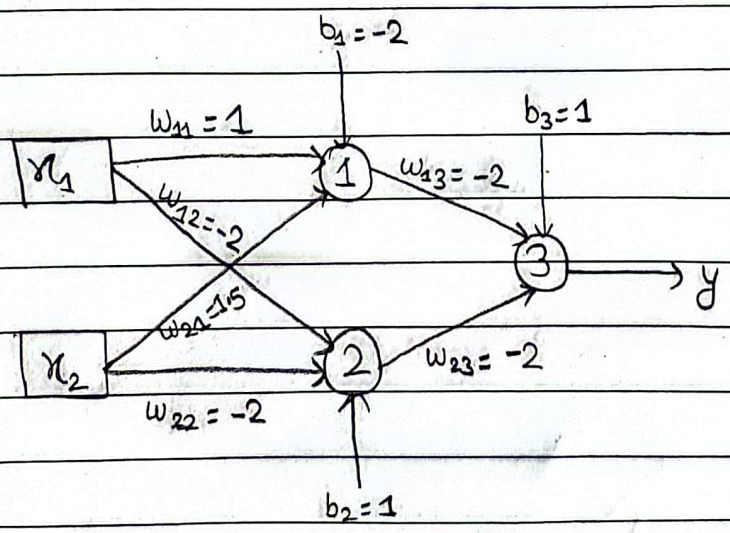


Fig : NOR Function

- XOR Function can be represented in terms of function of AND and NOR functions as below.

$$\text{NOR}(\text{AND}(x_1, x_2), \text{NOR}(x_1, x_2))$$

- Thus, Weights of MLP that acts as XOR function can be set as below:



CW

- Realize NOT and NAND function using perceptron.
- Can we realize XNOR function using perceptron? If yes, realize it using perceptron. Otherwise, realize XNOR function using MLP.

Heuristics For making the Backpropagation Algorithm Perform Better

There are methods that significantly improves the backpropagation algorithm's performance, as described below.

1. Stochastic vs. Batch Update:

As mentioned previously, the stochastic (sequential or online) mode of backpropagation learning is computationally faster than the batch mode for large training datasets. However, there can be difficulty in convergence in case of stochastic update. Therefore, better choice is to use mini-batch update strategy.

2. Maximizing Information Content:

Two ways of maximizing information content are:

- Use an example that results in the largest training error.
- Use an example that is radically different from all those previously used.

- In pattern - classification tasks using sequential backpropagation learning, a simple and commonly used technique is to randomize (i.e., shuffle) the order in which the examples are presented to the multilayer perceptron from one epoch to the next.

- Ideally, the randomization ensure that successive examples in an epoch presented to the network rarely belong to the same class.

Date
Page

3. Activation Function:

Sigmoid function and hyperbolic tangent function should be choice for activation function in MLP.

- Sigmoid activation function has following properties:

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

$$\Rightarrow x \rightarrow \varphi(x) \qquad \varphi(-x) = -\varphi(x)$$

- The function that holds above relation is called odd function. Thus, sigmoid functions are odd functions.

- We know that general form of hyperbolic tangent function is:

$$\varphi(x) = a\tanh(bx) = a * \frac{e^{bx} - e^{-bx}}{e^{bx} + e^{-bx}}$$

- According to LeCun (1993), $a = 1.7159$ and $b = 2/3 = 0.67$ suitable values for the above activation function.

- Thus hyperbolic tangent function has following useful properties:

$$\varphi(1) = 1 \qquad \varphi(-1) = -1 \qquad \varphi(0) = ab$$

4. Target Values:

It is important that the target values (desired response) be chosen within the range of the sigmoid activation function. For the limiting value 'a' and '-a', we set:

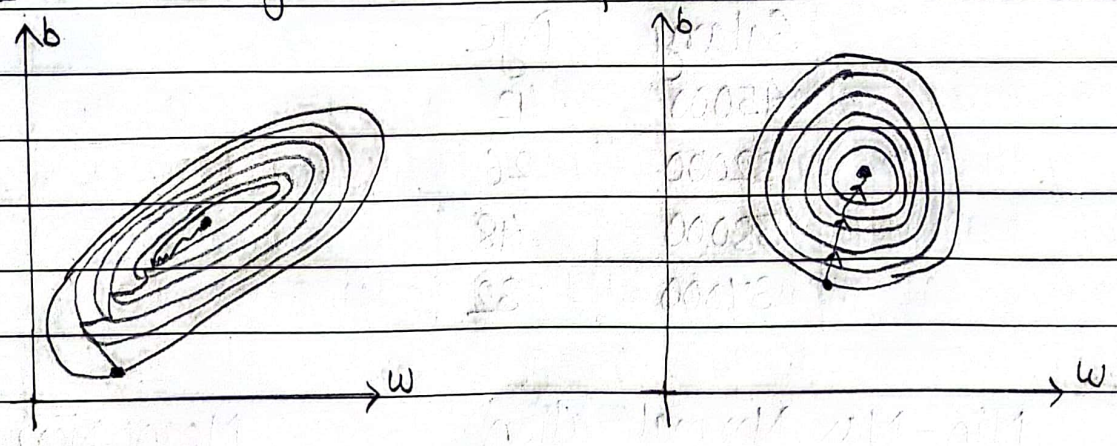
$$d_j = a - \varepsilon \qquad d_j = -a + \varepsilon$$

5. Normalizing the Inputs:

Each input variable should be preprocessed so that its mean value is close to zero. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values. It is required only when features have different ranges.

- For example, consider a data set containing two features, x_1 and x_2, where x_1 ranges from 0 - 100, while x_2 ranges from 10,000 - 200,000. While making prediction, the attribute x_2 will intrinsically influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor.

- Input features are of different scale (Unnormalized) leads to elongated loss function but normalized input features leads to more symmetric loss function.



- Thus with normalized input features gradient descent converges much faster.

Ways of Normalization

- Min-max Normalization / Scaling

$$x = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Mean Normalization / Scaling

$$x = \frac{x - \mu}{\max(x) - \min(x)}$$

- Standard Normalization / Scaling

$$x = \frac{x - \mu}{\sigma}$$

where, μ is mean

Example:

- Normalize the following data using Min-Max, mean, and Standard Normalization.

Salary	Age
45000	42
32000	26
58000	48
37000	32

Soln:

Min-Max Normalization

Salary	Age
0.50	0.73
0.00	0.00
1.00	1.00
1.19	0.27

Mean Normalization

Salary	Age
0.08	0.23
-0.42	-0.50
0.58	0.50
-0.23	-0.23

Standard Normalization

Salary	Age
0.20	0.59
-1.12	-1.29
1.53	1.29
-0.61	-0.59

Note:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}}$$

where, μ = mean and
N is number of data points.

6. Weight Initialization :

Good choice for the initial values of the synaptic weights and thresholds of the network can be of tremendous help in a successful network design.

- If weights are initialized with very high values, the activation field (weighted sum of inputs) becomes significantly higher and if an activation function like sigmoid is applied, the function maps its value near to 1 where the slope of gradient changes slowly and learning takes a lot of time. This problem is often referred to as the vanishing gradient.

- If weights are initialized with low values, activation field gets mapped near to 0. In such case, the backpropagation algorithm may operate on a very flat area around the origin of the error surface.

- For these reasons, the use of both large and small values for initializing the synaptic weights should be avoided.

- Thus, better weight initialization method is to initialize weights randomly such that weights have uniform distribution with zero mean and a variance equal to the reciprocal of the number of synaptic connections of a neuron. i.e.

$$\sigma_w = m^{-1/2} = \sqrt{\frac{1}{m}}$$

where, m is number of synaptic connections or links.

7. Learning Rate:

Learning rate should be between 0 and 1. It must be sufficiently small. Large value of learning rate may cause difficulty in convergence of algorithm.

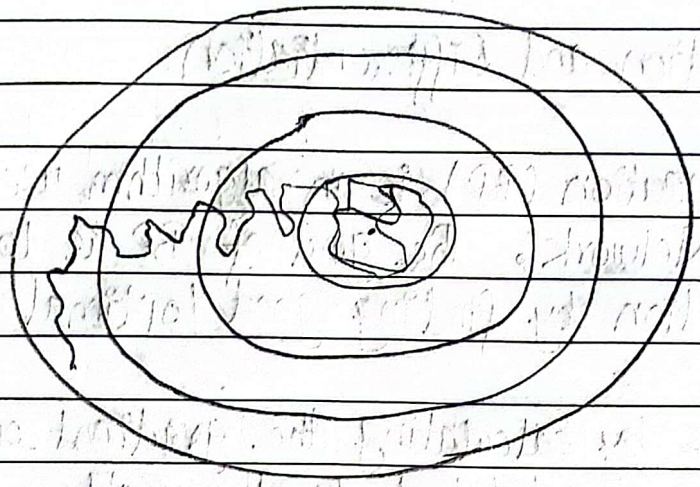
- The last layers usually have larger local gradients than the layers at the front end of the network. Hence, the learning-rate parameter should be assigned a smaller value in the last layers than in the front layers of the multilayer perceptron.

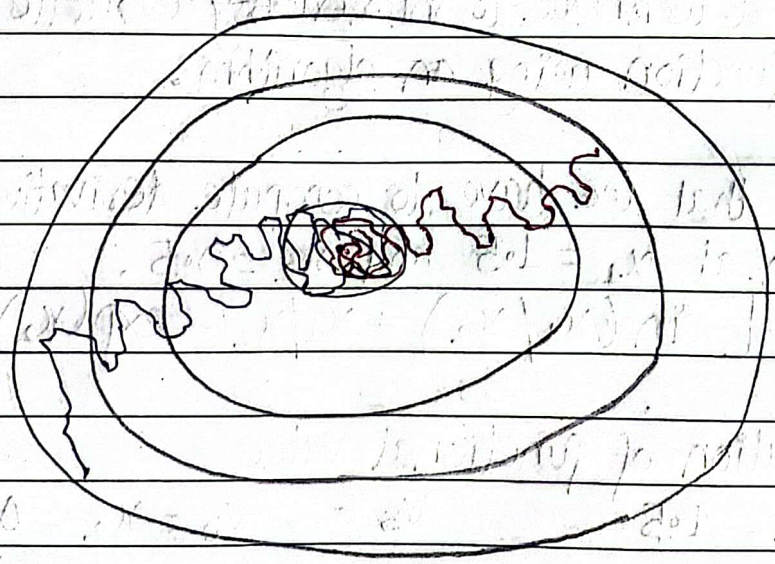
- Neurons with many inputs should have a smaller learning-rate parameter than neurons with few inputs so as to maintain a similar learning time for all neurons in the network.

- The learning rate needs to be reduced over time. The reason for this is that in the beginning, the bigger steps

can be taken but as the cost function reaches to its minimum, learning rate needs to be decreased.

- If the learning rate is same or large, the algorithm may never converge and will always oscillate around the minima (BLUE). If learning rate is decreased over time or sufficiently small, it will converge to minima in the later epochs (RED).





8. Learning From Hints:

Learning from examples is the process of taking input-output examples of an unknown function f and inferring an implementation of f. Learning from hints allows for general information about f to be used instead of just input-output examples.

Backpropagation and Differentiation

- The Backpropagation (BP) is an algorithm used to train the Neural Network. The goal of BP is to optimize the loss function by finding good / optimal weight values.
- The BP works by calculating the gradient of the loss function and propagated back to all weights. The BP uses Automatic Differentiation (AD) to calculate the gradient automatically.
- AD is a technique to numerically evaluate the derivative of a function using an algorithm.

- Suppose that we have to compute derivative of following function at $x_1 = 1.5$ and $x_2 = 0.5$.

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] \times [x_1/x_2 - \exp(x_2)]$$

- Computation of functional value

$V_a = x_1 = 1.5$ $\quad$ $V_b = x_2 = 0.5$

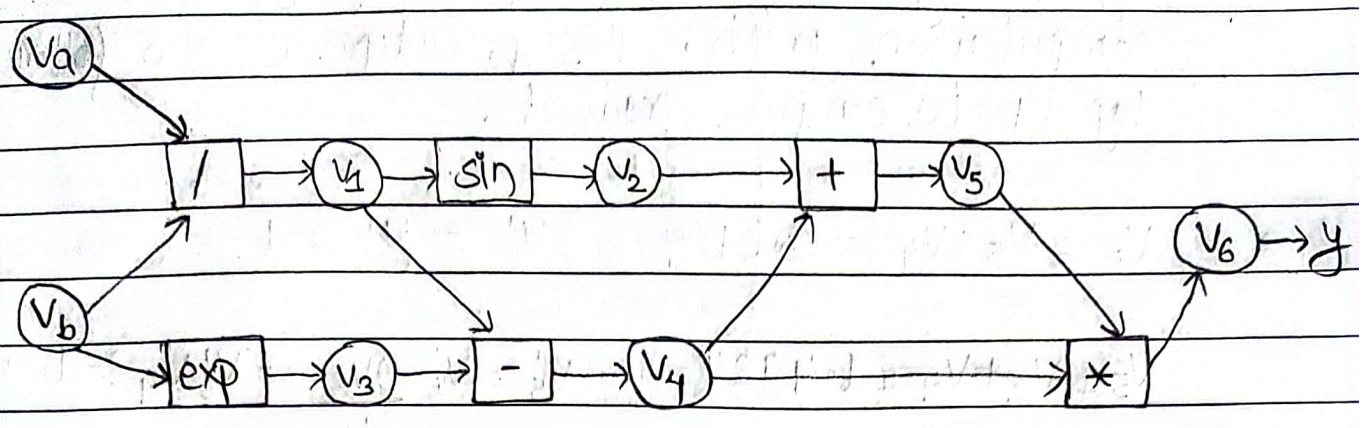
$V_1 = V_a / V_b = 3.0$ $\quad$ $V_2 = \sin(V_1) = 0.1411$

$V_3 = \exp(V_b) = 1.6487$ $\quad$ $V_4 = V_1 - V_3 = 1.4513$

$V_5 = V_2 + V_4 = 1.4924$ $\quad$ $V_6 = V_5 \times V_4 = 2.0167$

$y = V_6 = 2.0167$

- We have following computation graph of above computation.



- In case of forward mode AD we can compute derivatives as we go. Suppose we have to compute partial derivative w.r.t. x_1.

$V_a = x_1 = 1.5$	$V_a' = 1$
$V_b = x_2 = 0.5$	$V_b' = 0$
$V_1 = V_a / V_b = 3.0$	$V_1' = (V_b V_a' - V_a V_b') / V_b^2 = 2.0$
$V_2 = \sin(V_1) = 0.1411$	$V_2' = \cos(V_1) V_1' = -1.98$
$V_3 = \exp(V_b) = 1.6487$	$V_3' = \exp(V_b) . V_b' = 0$
$V_4 = V_1 - V_3 = 1.4513$	$V_4' = V_1' - V_3' = 2.0$
$V_5 = V_2 + V_4 = 1.4924$	$V_5' = V_2' + V_4' = 0.02$
$V_6 = V_5 \times V_4 = 2.0167$	$V_6' = y' = V_5 V_4' + V_4 V_5' = 3.0118$

- Reverse mode AD walks backwards through the computation graph and computes derivatives of the output with respect to the local variable. This quantity is sometimes called an adjoint variable $V_i' = \dfrac{\partial y}{\partial V_i}$

- Mathematically, the adjoint is computed by looking at adjoints of the children of vertex V_i on the graph:

$$V_i' = \sum_{j:\text{ children of } i} V_j' \frac{\partial V_j}{\partial V_i}$$

- In case of reverse AD we can compute derivatives by moving backward from output to input layer of computation graph. This procedure is also followed by BP to compute gradient.

$V_6 = V_5 \times V_4 = 2.0167 \qquad V_6' = y' = 1$

$V_5 = V_2 + V_4 = 1.4924 \qquad V_5' = V_6' \frac{\partial V_6}{\partial V_5} = V_6' V_4 = 1.3513$

$V_4 = V_1 - V_3 = 1.4513 \qquad V_4' = V_5' \frac{\partial V_5}{\partial V_4} + V_6' \frac{\partial V_6}{\partial V_4} = V_5' + V_6' V_5 = 2.8$

$V_3 = \exp(V_b) = 1.6487 \qquad V_3' = V_4' \frac{\partial V_4}{\partial V_3} = -V_4' = -2.8437$

$V_2 = \sin(V_1) = 0.1411 \qquad V_2' = V_5' \frac{\partial V_5}{\partial V_2} = V_5' = 1.3513$

$V_1 = V_a / V_b = 3.0 \qquad V_1' = V_2' \frac{\partial V_2}{\partial V_1} + V_4' \frac{\partial V_4}{\partial V_1} = V_2' \cos(V_1) + V_4' = 1.5059$

$V_b = x_2 = 0.5 \qquad V_b' = V_1' \frac{\partial V_1}{\partial V_b} + V_3' \frac{\partial V_3}{\partial V_b} = V_3' V_3 - V_1' V_1 / V_b = -13.7239$

$V_a = x_1 = 1.5 \qquad V_a' = V_1' \frac{\partial V_1}{\partial V_a} = V_1' / V_b = 3.0118$

Jacobian and Hessian

- The Jacobian is a matrix of all the first-order partial derivatives of a vector-valued function. In the neural network case, it is an N-by-W matrix, where N is the number of entries in our training set and W is the total number of parameters (weights) of our network.

- It can be generated by taking the partial derivatives of each output with respect to each weight.

- Each row of the Jacobian corresponds to a particular example in the training sample.

$$J = \begin{bmatrix} \frac{\partial F(x_1, w)}{\partial w_1} & \cdots & \frac{\partial F(x_1, w)}{\partial w_W} \\ \vdots & \ddots & \vdots \\ \frac{\partial F(x_N, w)}{\partial w_1} & \cdots & \frac{\partial F(x_N, w)}{\partial w_W} \end{bmatrix}$$

- While it is a good exercise to compute the gradient of a neural network with respect to a single parameter (e.g., a single weight), in practice this tends to be quite slow. This approach can't take advantage of vectorization. Thus, for vectorized implementation of backpropagation algorithm, it is necessary to use Jacobian.

- Hessian of a cost function is square matrix whose elements are second order partial derivative of the cost function with respect to each weight in the MLP.

$$H = \frac{\partial^2 f}{\partial w^2} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial w_n w_1} & \frac{\partial^2 f}{\partial w_n w_2} & \cdots & \frac{\partial^2 f}{\partial w_n^2} \end{bmatrix}$$

Extra

Role of Hessian in Online Learning

- The Hessian plays an important role in the study of neural networks:
  - The eigenvalues of the Hessian have a profound influence on the dynamics of back-propagation learning.
  - The inverse of the Hessian provides a basis for pruning insignificant synaptic weights from a MLP.
  - The Hessian is basic to the formulation of second-order optimization methods as an alternative to back-propagation learning.

- Typically, the Hessian of the error surface pertaining to a multilayer perceptron trained with the back-propagation algorithm has the following composition of eigenvalues:
  - a small number of small eigenvalues,
    a large number of medium-sized eigenvalues, and
  - a small number of large eigenvalues.

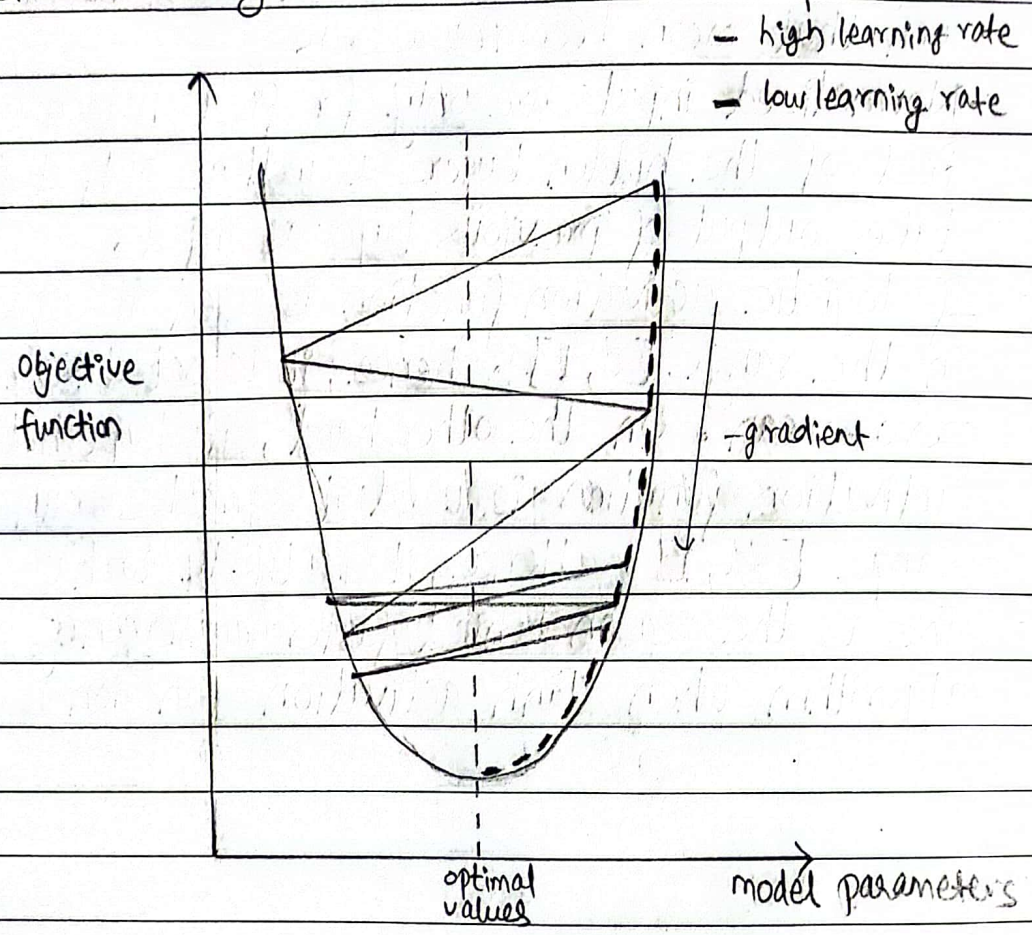
- It has been shown that optimal learning rate is inverse of largest eigenvalue of H.

$$\alpha_{opt} = \frac{1}{\lambda_{max}}$$

- Learning time of back propagation is sensitive to the ratio $\lambda_{max} / \lambda_{min}$, where λ_{max} and λ_{min} are largest and non-negative smallest eigenvalues of H.
- For the inputs with non-zero mean, ratio $\lambda_{max} / \lambda_{min}$ is large. This is the reason behind normalizing inputs such that mean becomes zero.
- Normalized inputs are only for first hidden layer of MLP. Rest of the hidden layers as well as output layer of MLP takes output of previous layer as input.
- If logistic activation function is used it squashes output in the range [0,1]. Hence, it is not likely to have zero-mean. On the other hand, if hyperbolic tangent activation function is used it squashes output in the range [-1,1]. Hence, it is likely to have zero-mean.
- This is the reason behind faster convergence of learning algorithm when Tanh activation function is used.

Learning Rate Annealing

- One of the key hyper parameters to set in order to train a neural network is the learning rate.
- If learning rate is set too low, training will progress very slowly as we are making very tiny updates to the weights in our network.
- However, if learning rate is set too high, it can cause undesirable divergent behavior in our loss function.



- One approach of using learning rate value properly is to start with large value of learning rate and reduce that the value of learning rate according to some predefined schedule. This approach is called learning rate annealing.

- Three commonly used learning rate annealing techniques or learning rate schedules are:
  - Time Based Decay, - Step Decay, - Exponential Decay

Date
Page

1. Time Based Decay

Mathematical formulation of time-based learning rate decay is given below :

$$\alpha = \frac{\alpha_0}{1 + \text{decay_rate} \times \text{epoch\#}}$$

Example :

Let $\alpha_0 = 0.2$ Decay_Rate = 1

For Epoch = 1 $\alpha = \frac{0.2}{1+(1\times1)} = \frac{0.2}{2} = 0.1$

For Epoch = 2 $\alpha = \frac{0.2}{1+(1\times2)} = \frac{0.2}{3} = 0.067$

For Epoch = 3 $\alpha = \frac{0.2}{1+(1\times3)} = \frac{0.2}{4} = 0.05$

So on.

2. Step Decay

Mathematical formulation of step decay of learning rate is given below. A typical way is to drop the learning rate by half after every 10 epochs.

$$\alpha = \alpha_0 \times \text{drop_rate}^{(\lfloor \text{epoch\#} / \text{epochs_drop} \rfloor)}$$

Example :

Let $\alpha_0 = 0.2$ drop_rate = 0.5 epochs_drop = 5

For Epoch = 5 $\alpha = 0.2 \times 0.5^{5/5} = 0.1$

For Epoch = 10 $\alpha = 0.2 \times 0.5^{10/5} = 0.05$

For Epoch = 15 $\alpha = 0.2 \times 0.5^{15/5} = 0.025$

So on

3. Exponential Decay

Mathematical formulation of exponential decay of learning rate is given below.

$$\alpha = \alpha_0 e^{(-k \times epoch\#)}$$

Example:

Let $\alpha_0 = 0.2$ $k = 0.1$

For Epoch = 1 $\alpha = 0.181$

For Epoch = 2 $\alpha = 0.164$

For Epoch = 3 $\alpha = 0.148$

So on.

Adaptive Control of Learning Rate

- Learning rate annealing techniques reduces learning rate equally for all parameters and applies the same learning rate with all parameters.

- However, in many practical situations, we need to change value of different parameters by different extent.

- Thus, the better approach is to use the techniques that gives learning rate a chance to adapt.

- The main idea behind adaptive learning rate is to use larger learning rate for updating parameters that are modified with small scale in past and use smaller learning rate for updating parameters that are modified with large scale in the past.

- Commonly used learning rate adaptation techniques are: Momentum, Adagrad, Adadelta, RMSProp, and Adam.

Momentum

- Momentum uses the past gradients to smooth out the parameter update.
- It computes an exponentially weighted average of gradients and then uses this gradient to update the weights.

$$V_t = \beta V_{t-1} + (1-\beta)\, dw_t$$

$$w_{t+1} = w_t - \alpha V_t$$

Adagrad

- In Adagrad, the variable v, called cache, keeps track of sum of the squared gradients, which in turn is used to normalize the parameter update.

$$V_t = V_{t-1} + dw_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t + \epsilon}}\, dw_t$$

- The effect of above equations is that weights receiving high gradients will have their effective learning rate reduced. On the other hand, weights receiving small gradients will have their effective learning rate increased.

Adadelta

- Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

RMSProp

- Adagrad decays the learning rate very aggressively (as the denominator grows). As a result, after a while, parameters receiving larger gradients will start receiving very small updates because of the decayed learning rate.
- To avoid this why not decay the denominator and prevent its rapid growth.
- RMSProp stands for root mean squared propagation and avoids monotonically increasing denominator of Adagrad.

- The central idea of RMSProp is keep the moving average of the squared gradients for each weight. And then divide the learning rate by root mean square of the squared gradients.

$$V_t = \beta V_{t-1} + (1-\beta)\, dw_t^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t + \epsilon}}\, dw_t$$

Adam

- Adam update can be considered as RMSProp with momentum. In place of raw and noisy gradient vector dw, weighted average of gradients are used.

Weighted average of gradients

$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\, dw_t$$

Weighted average of squared gradients

$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\, dw_t^2$$

<u>Bias Correction</u>

$$V_t = \frac{V_t}{(1-\beta_1^t)} \qquad S_t = \frac{S_t}{(1-\beta_2^t)}$$

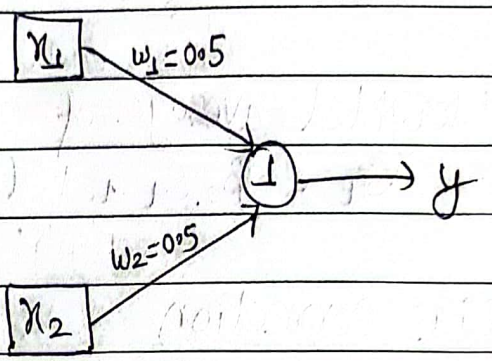
Finally, update weights as below:

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} V_t$$

Example:

- Consider following simple ANN with logistic activation function. Assume that $\alpha = 0.2$, $\beta = 0.8$, $\beta_1 = 0.8$, $\beta_2 = 0.8$. Calculate weight updates for the given training sample using (1) Adagrad (2) RMSProp (3) Adam and (4) Momentum.

x_1	x_2	t
1	0.6	0.8
0.5	0.5	0.6
0.2	0.8	0.3



Sol:

1) For Adagrad

For training example (1, 0.6, 0.8):

$$y_{in} = \overset{w_1x_1 + w_2x_2}{0.5 * 1 + 0.5 * 0.6} = 0.8$$

$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6899$$

$$dw_1 = (y - t) \times y \times (1 - y) \times x_1$$
$$= (0.6899 - 0.8) \times 0.6899 \times (1 - 0.6899) \times 1$$
$$= -0.023545$$

$$dw_2 = (y - t) \times y \times (1 - y) \times x_2$$
$$= (0.6899 - 0.8) \times 0.6899 \times (1 - 0.6899) \times 0.6$$
$$= -0.01413$$

$(V_t = V_{t-1} + dw_t^2)$

$$V_1 = -0.02355^2 = 0.000554$$
$$V_2 = -0.01413^2 = 0.000199$$

$$\left(w_{t+1} = w_t - \frac{\alpha}{\sqrt{V_t + \epsilon}} dw_t \right)$$

$$w_1 = \overset{0.5}{w_1} - \frac{0.2}{\sqrt{0.000554}} \times (-0.02355) = 0.7$$

$$w_2 = \overset{w_2}{0.5} - \frac{0.2}{\sqrt{0.000199}} \times (-0.01413) = 0.7$$

<u>For training example (0.5, 0.5, 0.6) :</u>

$$y_{in} = \overset{w_1 x_1 + w_2 x_2}{0.7 * 0.5 + 0.7 * 0.5} = 0.7$$

$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6682$$

$$dw_1 = (y-t) \times y \times (1-y) \times x_1$$
$$= (0.6682 - 0.6) \times 0.6682 \times (1 - 0.6682) \times 0.5$$
$$= 0.007559$$

$$dw_2 = (y-t) \times y \times (1-y) \times x_2$$
$$= (0.6682 - 0.6) \times 0.6682 \times (1 - 0.6682) \times 0.5$$
$$= 0.007559$$

$$V_1 = 0.000554 + 0.007559^2 = 0.000611$$
$$V_2 = 0.000199 + 0.007559^2 = 0.000257$$

$$w_1 = 0.7 - \frac{0.2}{\sqrt{0.000611}} \times 0.007559 = 0.639$$

$$w_2 = 0.7 - \frac{0.2}{\sqrt{0.000257}} \times 0.007559 = 0.606$$

<u>For training example (0.2, 0.8, 0.3);</u>

$$y_{in} = 0.639 * 0.2 + 0.606 * 0.8 = 0.6126$$

$$y = \frac{1}{1 + \exp(-y_{in})} \doteq 0.6485$$

$$dw_1 = (0.6485 - 0.3) \times 0.6485 \times (1 - 0.6485) \times 0.2$$
$$= 0.0159$$

$$dw_2 = (0.6485 - 0.3) \times 0.6485 \times (1 - 0.6485) \times 0.8$$
$$= 0.0636$$

$$v_1 = 0.000611 + 0.0159^2 \doteq 0.00084$$
$$v_2 = 0.000257 + 0.0636^2 = 0.00430$$

$$w_1 = 0.639 - \frac{0.2}{\sqrt{0.00084}} \times 0.0159 \doteq 0.529$$

$$w_2 = 0.606 - \frac{0.2}{\sqrt{0.00430}} \times 0.0636 = 0.412$$

2) For RMSProp

For Training Example (1, 0.6, 0.8):

$y_{in} = 0.5*1 + 0.5*0.6 = 0.8$

$$y = \frac{1}{1+\exp(-y_{in})} = 0.6899$$

$dw_1 = (y-t) \times y \times (1-y) \times x_1$
$= (0.6899 - 0.8) \times 0.6899 \times (1 - 0.6899) \times 1$
$= -0.02355$

$dw_2 = (y-t) \times y \times (1-y) \times x_2$
$= (0.6899 - 0.8) \times 0.6899 \times (1 - 0.6899) \times 0.6$
$= -0.01413$

$(V_t = \beta V_{t-1} + (1-\beta) dw_t^2)$

$V_1 = 0.8*0 + (1-0.8)*(-0.02355)^2 = 0.000111$

$V_2 = 0.8*0 + (1-0.8)*(-0.01413)^2 = 0.0000399$

$$w_1 = \overset{0.5}{w_1} - \frac{0.2}{\sqrt{0.000111}} \times (-0.02355) = 0.947$$

$$w_2 = \overset{0.5}{w_2} - \frac{0.2}{\sqrt{0.0000399}} \times (-0.01413) = 0.947$$

For Training Example (0.5, 0.5, 0.6):

$y_{in} = 0.947*0.5 + 0.947*0.5$
$= 0.947.$

$$y = \frac{1}{1+\exp(-y_{in})} = 0.7205$$

$$dw_1 = (0.7205 - 0.6) * 0.7205 * (1 - 0.7205) * 0.5$$
$$= 0.0121$$
$$dw_2 = (0.7205 - 0.6) * 0.7205 * (1 - 0.7205) * 0.5$$
$$= 0.0121$$

$$V_1 = 0.8 * 0.000111 + (1 - 0.8) * (0.121)^2$$
$$= 0.003017$$
$$V_2 = 0.8 * 0.0000399 + (1 - 0.8) * (0.121)^2$$
$$= 0.002960$$

$$w_1 = 0.947 - \frac{0.2}{\sqrt{0.003017}} * (0.0121)$$
$$= 0.90294$$
$$w_2 = 0.947 - \frac{0.2}{\sqrt{0.002960}} * (0.0121)$$
$$= 0.90252.$$

<u>For Training Example (0.2, 0.8, 0.3):</u>

$$y_{in} = 0.90294 * 0.2 + 0.90252 * 0.8$$
$$= 0.903604$$
$$y = \frac{1}{1 + \exp(-y_{in})} = 0.7115$$
$$dw_1 = (0.7115 - 0.3) * 0.7115 * (1 - 0.7115) * 0.2 = 0.0169$$
$$dw_2 = (0.7115 - 0.3) * 0.7115 * (1 - 0.7115) * 0.8 = 0.0676$$

$$V_1 = 0.8 * 0.003017 + (1 - 0.8) * (0.0169)^2 = 0.00247$$
$$V_2 = 0.8 * 0.002960 + (1 - 0.8) * (0.0676)^2 = 0.00328$$

$$w_1 = 0.90294 - \frac{0.2}{\sqrt{0.00247}} * (0.0169) = 0.8349$$
$$w_2 = 0.90252 - \frac{0.2}{\sqrt{0.00328}} * (0.0676) = 0.6665$$

3) For Adam

For training example (1, 0.6, 0.8);

$$y_{in} = 0.5 * 1 + 0.5 * 0.6 = 0.8$$

$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6899$$

$$dw_1 = (0.6899 - 0.8) * 0.6899 * (1 - 0.6899) * 1 = -0.02355$$

$$dw_2 = (0.6899 - 0.8) * 0.6899 * (1 - 0.6899) * 0.6 = -0.01413$$

$$\{ V_t = \beta_1 V_{t-1} + (1 - \beta_1) dw_t \}$$

$$V_1 = 0.8 * 0 + (1 - 0.8) * (-0.02355) = -0.00471$$

$$V_2 = 0.8 * 0 + (1 - 0.8) * (-0.01413) = -0.002826$$

$$\{ S_t = \beta_2 S_{t-1} + (1 - \beta_2) dw_t^2$$

$$S_1 = 0.8 * 0 + (1 - 0.8) * (-0.02355)^2 = \cancel{0.0000399}\ 0.0001109$$

$$S_2 = 0.8 * 0 + (1 - 0.8) * (-0.01413)^2 = 0.0000399$$

$$\{ w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} V_t \}$$

$$w_1 = 0.5 - \frac{0.2}{\sqrt{0.0001109}} * (-0.00471) = 0.58945$$

$$w_2 = 0.5 - \frac{0.2}{\sqrt{0.0000399}} * (-0.002826) = 0.58947$$

For training example (0.5, 0.5, 0.6);

$$y_{in} = 0.58945 * 0.5 + 0.58947 * 0.5$$
$$= 0.58946$$

$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6432$$

$$dw_1 = (0.6432 - 0.6) * 0.6432 * (1 - 0.6432) * 0.5 = 0.00496$$
$$dw_2 = (0.6432 - 0.6) * 0.6432 * (1 - 0.6432) * 0.5 = 0.00496$$

$$V_1 = 0.8 * (-0.00471) + (1 - 0.8) * 0.00496 = -0.002776$$
$$V_2 = 0.8 * (-0.002826) + (1 - 0.8) * 0.00496 = -0.001269$$

$$S_1 = 0.8 * 0.0001109 + (1 - 0.8) * (0.00496)^2 = 0.0000936$$
$$S_2 = 0.8 * 0.0000399 + (1 - 0.8) * (0.00496)^2 = 0.0000368$$

$$w_1 = 0.58945 - \frac{0.2}{\sqrt{0.0000936}} * (-0.002776) = 0.64684$$
$$w_2 = 0.58947 - \frac{0.2}{\sqrt{0.0000368}} * (-0.001269) = 0.63131$$

<u>For training example (0.2, 0.8, 0.3):</u>

$$y_{in} = 0.64684 * 0.2 + 0.63131 * 0.8 = 0.63442$$
$$y = \frac{1}{1 + \exp(-y_{in})} = 0.6535$$
$$dw_1 = (0.6535 - 0.3) * 0.6535 * (1 - 0.6535) * 0.2 = 0.016$$
$$dw_2 = (0.6535 - 0.3) * 0.6535 * (1 - 0.6535) * 0.8 = 0.064$$

$$V_1 = 0.8 * (-0.002776) + (1 - 0.8) * 0.016 = 0.0009792$$
$$V_2 = 0.8 * (-0.001269) + (1 - 0.8) * 0.064 = 0.0117848$$

$$S_1 = 0.8 * 0.0000936 + (1 - 0.8) * (0.016)^2 = 0.000126$$
$$S_2 = 0.8 * 0.0000368 + (1 - 0.8) * (0.064)^2 = 0.000849$$

$$w_1 = 0.64684 - \frac{0.2}{\sqrt{0.000126}} * 0.0009792 = 0.6294$$
$$w_2 = 0.63131 - \frac{0.2}{\sqrt{0.000849}} * 0.0117848 = 0.5504$$

4) For Momentum

For training example (1, 0.6, 0.8);

$y_{in} = 0.5 * 1 + 0.5 * 0.6 = 0.8$

$y = \frac{1}{1 + \exp(-y_{in})} = 0.6899$

$dw_1 = (0.6899 - 0.8) * 0.6899 * (1 - 0.6899) * 1 = -0.02355$

$dw_2 = (0.6899 - 0.8) * 0.6899 * (1 - 0.6899) * 0.6 = -0.01413$

$\{V_t = \beta V_{t-1} + (1-\beta)\, dw_t\}$

$V_1 = 0.8 * 0 + (1 - 0.8) * (-0.02355) = -0.00471$

$V_2 = 0.8 * 0 + (1 - 0.8) * (-0.01413) = -0.002826$

$\{w_{t+1} = w_t - \alpha V_t\}$

$w_1 = 0.5 - 0.2 * (-0.00471) = 0.500942$

$w_2 = 0.5 - 0.2 * (-0.002826) = 0.5005652$

For training example (0.5, 0.5, 0.6);

$y_{in} = 0.500942 * 0.5 + 0.5005652 * 0.5 = 0.5007536$

$y = \frac{1}{1 + \exp(-y_{in})} = 0.6226$

$dw_1 = (0.6226 - 0.6) * 0.6226 * (1 - 0.6226) * 0.5 = 0.00266$

$dw_2 = (0.6226 - 0.6) * 0.6226 * (1 - 0.6226) * 0.5 = 0.00266$

$V_1 = 0.8 * (-0.00471) + (1 - 0.8) * 0.00266 = -0.003236$

$V_2 = 0.8 * (-0.002826) + (1 - 0.8) * 0.00266 = -0.001729$

$w_1 = 0.500942 - 0.2 * (-0.003236) = 0.50159$

$w_2 = 0.5005652 - 0.2 * (-0.001729) = 0.500911$

For training example (0.2, 0.8, 0.3);

$y_{in} = 0.50159 * 0.2 + 0.500911 * 0.8 = 0.50105$

$y = \dfrac{1}{1 + \exp(-y_{in})} = 0.6227$

$dw_1 = (0.6227 - 0.3) * 0.6227 * (1 - 0.6227) * 0.2 = 0.0152$

$dw_2 = (0.6227 - 0.3) * 0.6227 * (1 - 0.6227) * 0.8 = 0.0607$

$V_1 = 0.8 * (-0.003236) + (1 - 0.8) * 0.0152 = 0.0004512$

$V_2 = 0.8 * (-0.001729) + (1 - 0.8) * 0.0607 = 0.0107568$

$w_1 = 0.50159 - 0.2 * 0.0004512 = 0.5015$

$w_2 = 0.500911 - 0.2 * 0.0107568 = 0.4988$

Example: same as before bt.

Note: Use bias correction feature in each case.

Date
Page

Cross-Validation

- In machine learning, datasets are divided into three sets: Training set, Validation set, and Test set. Split ratio of dataset into three sets is not fix. It depends upon size of the dataset. If we have very large dataset split ratio can be 98:1:1. Otherwise, we can opt for split ratio 80:10:10 or 70:15:15 so on.

- Training set is used to train neural network and update weights on the basis of error. Validation set is used to compute prediction error using weights resulted from training. If the error is higher than a user-defined threshold then the whole training validation epoch is repeated.

- The testing set is then used to measure the performance of the network. This is the data that was never used throughout the training and validation phase.

- Cross-validation is a re-sampling procedure used to evaluate machine learning models on a limited data sample.

- K-Fold cross-Validation (Multi-Fold Cross validation) is widely used for cross validation. If we choose K=10, it becomes 10-fold cross validation.

- In this approach, the original sample is randomly partitioned into K equal sized subsamples. Out of the K subsamples, a single subsample is used as the testing set, and the remaining K-1 subsamples are used as training + validation data.

- The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the testing data. The k results can then be averaged to produce a single estimation.

- The advantage of this method is that all observations are used for both training and testing, and each observation is used for testing exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter.

- For example, setting $k=2$ results in 2-fold cross-validation. In 2-fold cross-validation, we randomly shuffle the dataset into two sets d_0 and d_1, so that both sets are equal size. We then train on d_0 and test on d_1, followed by training on d_1 and testing on d_0.

- When $k=n$ (the number of observations), k-fold cross-validation is equivalent to leave-one-out cross-validation. In this case, $n-1$ examples are used to train the model, and the model is validated by testing it on the example that is left out. This approach is suitable when we have small number of training example.

- To make the cross-validation procedure concrete, let's look at a worked example. Imagine we have a data sample with 6 observations:

  $\{(1,3), (2,5), (3,7), (4,9), (5,11), (6,13)\}$

- Let $k=3$. That means we will shuffle the data and then split the data into 3 groups.

Fold 1 = { (1,3), (5,11) }
Fold 2 = { (6,13), (3,7) }
Fold 3 = { (2,5), (4,9) }

- Three models are trained and evaluated with each fold given a chance to be the held out test set.
For example :-
  - Model 1 : Trained on Fold 1 + Fold 2, Tested on Fold 3
  - Model 2 : Trained on Fold 2 + Fold 3, Tested on Fold 1
  - Model 3 : Trained on Fold 1 + Fold 3, Tested on Fold 2

- ~~When k=m~~
- Holdout method is alternative to K-Fold Cross-validation. In this approach, we randomly divide data points to two sets d_0 and d_1, usually called the training set and the test set, respectively. Usually, the test set is smaller than the training set. We then train on d_0 and test on d_1.

- In typical cross-validation, results of multiple runs of model-testing are averaged together. In contrast, the holdout method, involves a single run.

- It should be used with caution because without such averaging of multiple runs, one may achieve highly misleading results. Predictive accuracy will tend to be unstable since it will not be smoothed out by multiple iterations.

Convolutional Neural Network (CNN):

- CNN is a type of multilayer neural network specially designed for detecting, recognizing, and classifying images.

- Every CNN consist following three types of layers: Convolution Layer, Pooling Layer, and Fully-connected Layer.

Typical architecture of CNN is shown below:

Convolution Layer

- Convolution layer extracts features from an input image. It preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter ~~or~~ or kernel and produces an image with convolved features.

- Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3x3 as shown in below.

The matrix formed by sliding the filter over the image and

computing the dot product is called Feature Map or Convolved Feature.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5x5- Image Matrix

1	0	1
0	1	0
1	0	1

3x3- Filter Matrix

$\Longrightarrow$

4	3	4
2	4	3
2	3	4

Feature Map

Image matrix Filter Matrix

$$\begin{bmatrix}1&1&1\\0&1&1\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+1+0+1+0+0+0+1 = 4$$

$$\begin{bmatrix}1&1&0\\1&1&1\\0&1&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+0+0+1+0+0+0+1 = 3$$

$$\begin{bmatrix}1&0&0\\1&1&0\\1&1&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+0+0+1+0+1+0+1 = 4$$

$$\begin{bmatrix}0&1&1\\0&0&1\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 0+0+1+0+0+0+0+0+1 = 2$$

$$\begin{bmatrix}1&1&1\\0&1&1\\0&1&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+1+0+1+0+0+0+1 = 4$$

$$\begin{bmatrix}1&1&0\\1&1&1\\1&1&0\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+0+0+1+0+1+0+0 = 3$$

$$\begin{bmatrix}0&0&1\\0&0&1\\0&1&1\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 0+0+1+0+0+0+0+0+1 = 2$$

$$\begin{bmatrix}0&1&1\\0&1&1\\1&1&0\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 0+0+1+0+1+0+1+0+0 = 3$$

$$\begin{bmatrix}1&1&1\\1&1&0\\1&0&0\end{bmatrix}\cdot\begin{bmatrix}1&0&1\\0&1&0\\1&0&1\end{bmatrix} = 1+0+1+0+1+0+1+0+0 = 4$$

Feature Map

- Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

Pooling Layer

- Spatial Pooling or simply Pooling reduces the dimensionality of each feature map but retains the most important information.
- It is also called sub-sampling or down-sampling. Spatial Pooling can be of different types: Max, Average, Sum, etc.
- In case of Max Pooling, we define a spatial neighborhood (for example, a 3x3 window) and take the largest element from the rectified feature map within that window.
- Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

$\Rightarrow$

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Fig: 3x3 Max Pooling over 5x5 convolved feature

Fully Connected Layer

- The Fully Connected Layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer. The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer.
- The output from the Convolutional and Pooling layers represent high-level features of the input image. The purpose of the fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.
- The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.
- The Fully-Connected layer learns non-linear function in output produced by Convolutional and Pooling Layer.
- We should flatten the image into a column vector. The flattened output is fed to the Fully Connected Layer and backpropagation can be applied to train the network.

Fast Learning Method : Newton's Method

- One of the main problem with gradient descent is slow convergence.
- Newton's method is a second-order algorithm because it makes use of the Hessian matrix.
- This method's objective is to find better training directions by using the second derivatives of the loss function.
- Second order Taylor series expansion of the cost function around point $w(n)$ is given as below.

$$\Delta F(w(n)) = F(w(n+1)) - F(w(n))$$
$$= g^T(n)\, w(n) + \frac{1}{2} \Delta w^T(n)\, H(n)\, w(n) \quad \text{--- (1)}$$

- Where, g is gradient vector of cost function evaluated at $w(n)$ and $H(n)$ is hessian evaluated at $w(n)$.

- Differentiating eq (1) w.r.t $w(n)$ and equating it with zero, we get.

$$g(n) + w(n)\, H(n) = 0$$
$$w(n) = -g(n)\, H^{-1}(n) \quad \text{--- (2)}$$

- Since, $\Delta w(n) = w(n+1) - w(n)$

$$\Rightarrow w(n+1) = w(n) + \Delta w(n)$$
$$= w(n) - g(n)\, H^{-1}(n) \quad \text{--- (3)}$$

- This equation is used to update weights of neural network.
- Newton's method converges quickly and does not exhibit the zigzagging behaviour that sometimes characterizes the method of gradient descent.
- However, for Newton's method to work, the Hessian $H(n)$ has to be a positive definite matrix for all n.
- Unfortunately, in general, there is no guarantee that $H(n)$ is positive definite at every iteration of the algorithm.

- In any event, a major limitation of Newton's method is its computational complexity.

Fast Learning Method : Conjugate Gradient

- The Backpropagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient).
- This is the direction in which the cost function decreases most rapidly.
- Although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence due to zigzag nature of convergence path.
- If gradient descent finds minimum in each search direction, successive search directions will be orthogonal to each other. This is the reason behind zig-zag convergence path of the gradient descent.

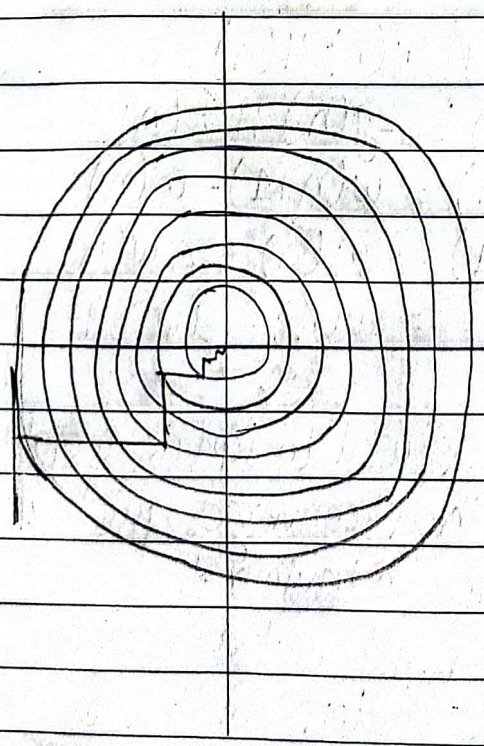


Fig: Convergence of Gradient Descent

- In Backpropagation, a learning rate is used to determine the length of the weight update (step size).
- However, In the conjugate gradient algorithms, the step size is adjusted at each iteration. A search is made along the conjugate gradient direction to determine the step size, which minimizes the cost function along that line.
- The concept behind conjugate gradient descent is that since each step is taking the same general direction, the combination of search directions should minimize the zig-zag nature of convergence path and capture the general trend towards the minimum.

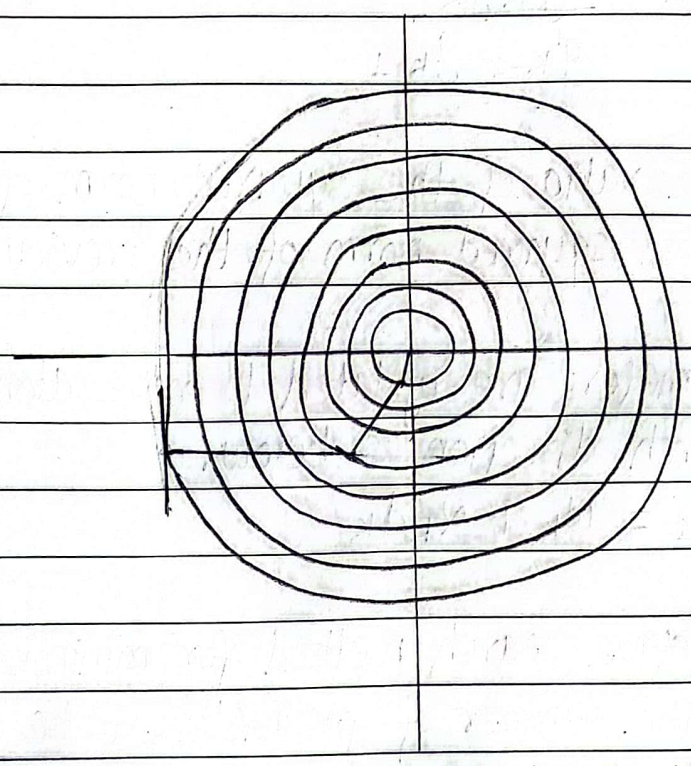


Fig: Convergence of Conjugate Gradient

- All of the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$d_0 = -g_0$$

- Then the next search direction is determined so that it is conjugate to previous search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction.

$$d_k = -g_k + \beta_k d_{k-1}$$

- Here β is called the conjugate parameter, and there are different ways to calculate it. Using the Fletcher-Reeves procedure β can be calculated as below.

$$\beta_k = \frac{g_k^T g_k}{g_{k-1}^T \cdot g_{k-1}}$$

This is the ratio of the squared norm of the current gradient to the squared norm of the previous gradient.

- Finally, parameters are updated to move along the current search direction as below.

$$w_{k+1} = w_k + \alpha_k d_k$$

- It is an effective search method for minimizing objective functions if the number of parameters to be optimized are few.

Auto associative Neural Network :

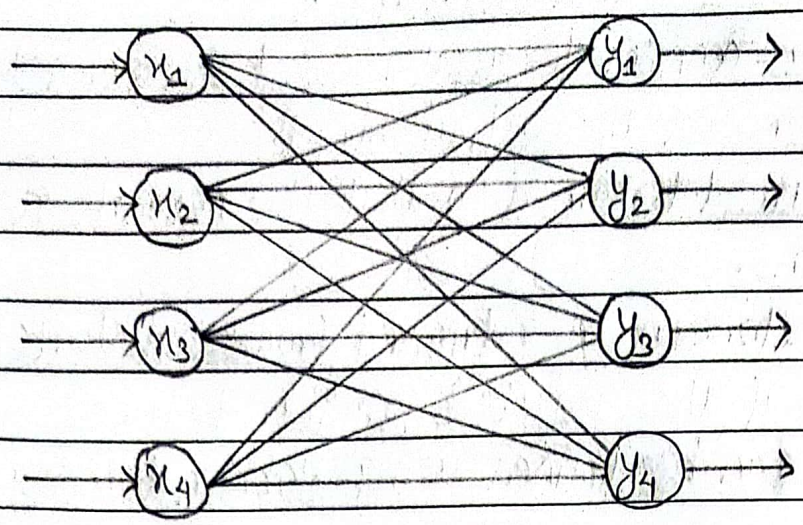
- This is a fully-connected neural network in which the input training vector and the output target vectors are the same.
- The weights are determined so that the network stores a set of patterns.
- A stored vector can be retrieved from a distorted or noisy vector if the input is similar to it.
- This network can be trained using Hebbian learning rule or Heb's rule.
- The rule is based on the assumption that if two neighbor neurons activated and deactivated at the same time. Then the weight connecting these neurons should increase. For neurons operating in the opposite phase, the weight between them should decrease. If there is no signal correlation, the weight should not change.
- This assumption can be described by using following formula.

$$\Delta w_{ij} = \alpha x_i y_j$$

where, w_{ij} is weight from neuron i to neuron j
α is learning rate
x_i is input
y_j is output

- Activation function for auto associative neural network is

$y_j = 1$ if $y_{in} > C$,
$y_j = x_j$ if $y_{in} = C$ and
$y_j = -1$ if $y_{in} < C$



Architecture of AANN

Training Algorithm

1. Initialize all weights and bias to zero.
2. For each vector to be stored (Say s) perform steps 3 to 5
3. For each input unit i, set $x_i = s_i$
4. For each output unit j, set $y_j = s_j$
5. Adapt weights as : $w_{ij} = w_{ij} + \alpha x_i y_j$

Weights can also be determined as : $w = s^T \cdot s$

Example:

Use the Heb's rule to store the pattern [1, 1, -1, -1] in an AANN. Find weight matrix and Test the input vector [-1, 1, -1, -1].

Sol^n:

Here, pattern to be stored (s) = [1, 1, -1, -1].

$$\text{Now, } w = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Now,

Test input = [-1 1 -1 -1]

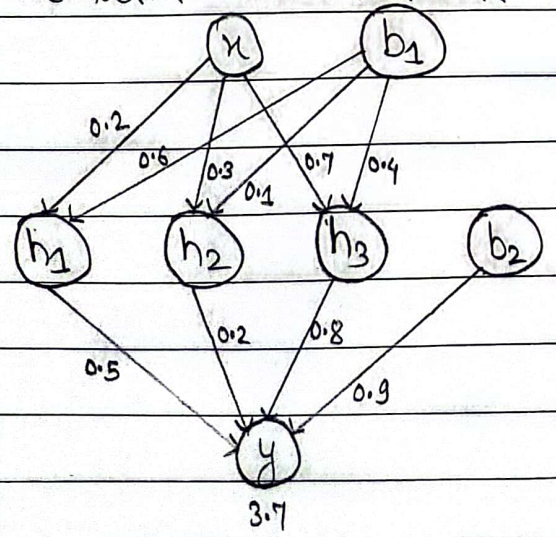
The calculate $y_{in} = x \cdot w = [2\ 2\ -2\ -2]$

Finally compute $y = [1\ 1\ -1\ 1]$

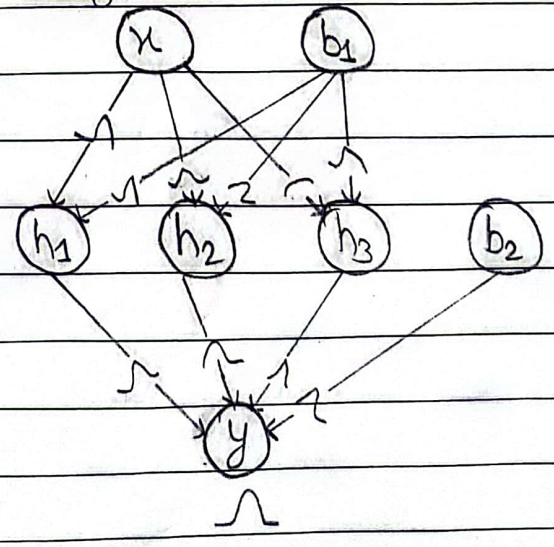
Bayesian Neural Network

- Taking a probablistic approach to neural network allows to account for uncertainty, so that models can assign less levels of confidence to incorrect predictions.
- Sources of uncertainty can be found in the data, due to measurement error or noise in the labels, or the model, due to insufficient data availability for the model to learn effectively.
- The idea behind Bayesian neural network is that, instead of learning specific weight (and bias) values in the neural network, the Bayesian approach learns weight distributions - from which we can sample to produce an output for a given input - to encode weight uncertainty.

Standard Neural Network



Bayesian Neural Network



UNIT 4:

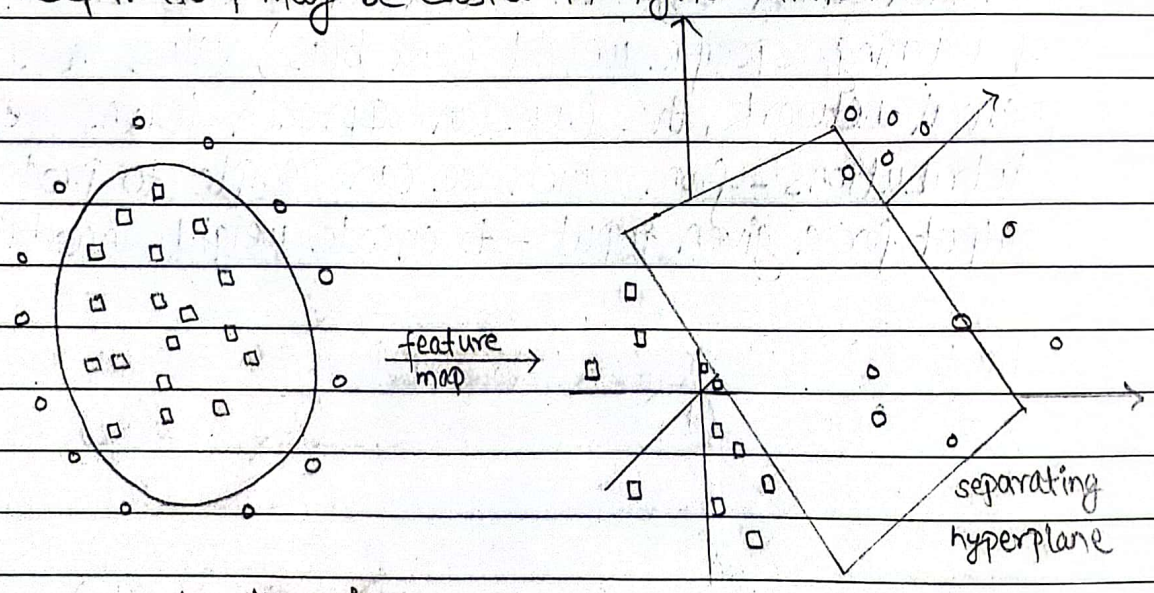
Radial Basis Function Networks

Kernel Methods

- Kernels or kernel methods (also called kernel functions) are sets of different types of algorithms that are being used for pattern analysis. They are used to solve a non-linear problem by using a linear classifier.

- Sometimes it may be difficult to divide lower dimension dataset using a linear line or hyperplane. Here comes the use of kernel function which takes the points to higher dimensions, solves the problem over there and returns the output.

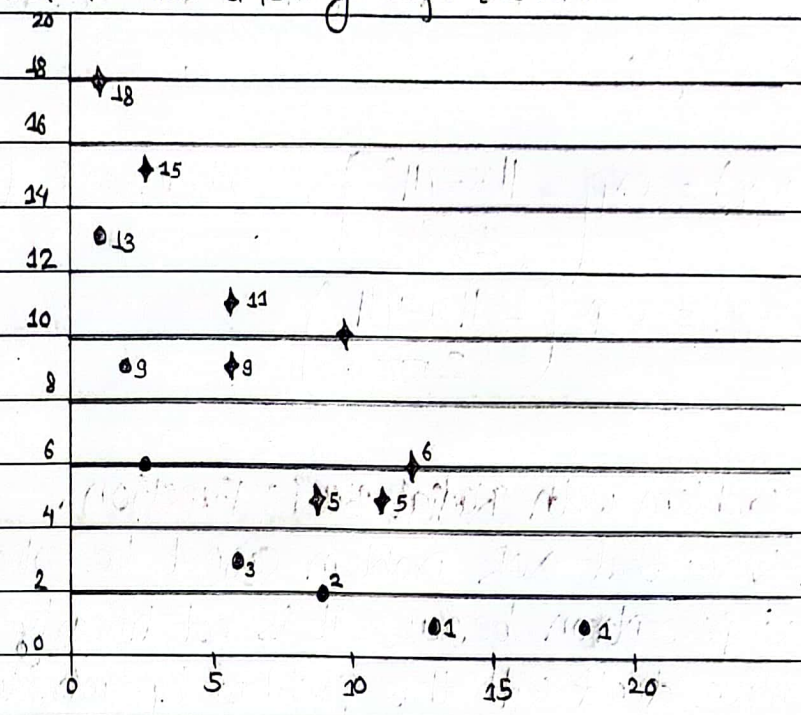
- Consider the situation shown in the figure given.
  Separation may be easier in higher dimensions



complex in low dimensions

simple in higher dimensions

- Consider the following data points : { (1,18), (1,13), (2,9), (3,6), (3,15), (6,11), (6,9), (6,3), (9,5), (9,2), (10,10), (11,5), (12,6), (13,1), (16,3), (18,1)}
- If plot the above dataset in graph, we can clearly see that the dataset is not linearly separable.



page-5

- There are different types of kernel like linear kernel, polynomial kernel, exponential kernel, Gaussian kernel, etc.
- Gaussian kernel and exponential kernel are examples of a radial basis function kernel. The equations given below are Gaussian and exponential kernel functions respectively.

aussian ernel $k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$, where σ is free parameter

ponential rnel function $k(x,y) = \exp\left(-\frac{\|x-y\|}{2\sigma^2}\right)$

XOR Problem with Radial Basis Function.

- We know that XOR problem cannot be solved by single layer perceptron because it is not linearly separable.
- We have dealt with this problem by introducing a hidden layer with two neurons in MLP.
- Alternatively, we can deal with XOR problem using Radial Basis Function (RBF).
- To achieve the XOR function we would like to construct a pattern classifier that produces the output 0 for the input patterns (0,0), and (1,1) and output 1 for the input patterns (0,1) and (1,0).

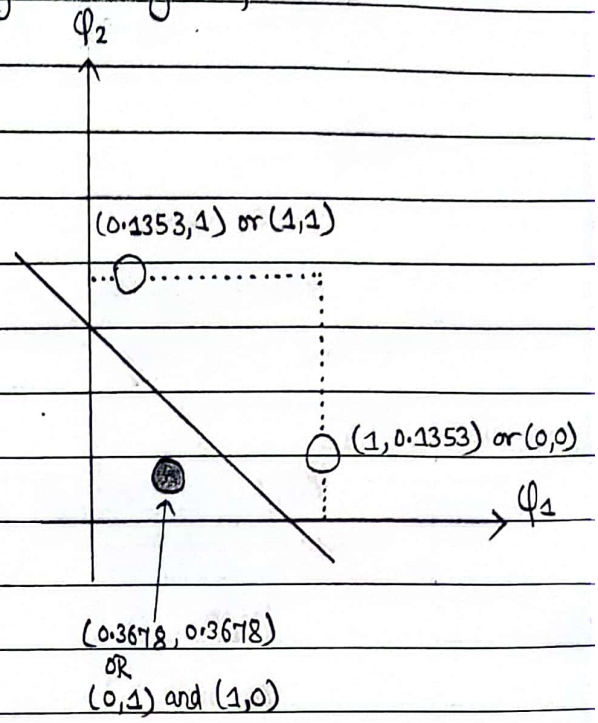
- Lets define a pair of Gaussian hidden functions as below:

$$\varphi_1(x) = \exp(-\|x-c_1\|^2) \qquad \varphi_2(x) = \exp(-\|x-c_2\|^2)$$

- RBF centers need to be chosen, assume that $c_1 = (0,0)$ and $c_2 = (1,1)$
- After applying radial basis function on the XOR input patterns, we will get the transformed input patterns as shown below.

- If we draw the transformed data points we will get the graph as below, which is clearly linearly separable.

x_1	x_2	φ_1	φ_2
0	0	1.0	0.1353
0	1	0.3678	0.3678
1	0	0.3678	0.3678
1	1	0.1353	1.0



Exercise:

- Solve the XOR problem using RBF by taking (1,0) and (0,1) as centers.
- Solve the XOR problem using RBF such that transform 2D data points into 4D data points by taking each of the data point as center. (Hint use the RBF $\varphi(x) = \exp(-\|x-c\|^2)/2$.

Radial Basis Function Networks

- Radial basis function (RBF) networks are artificial neural networks that typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer.

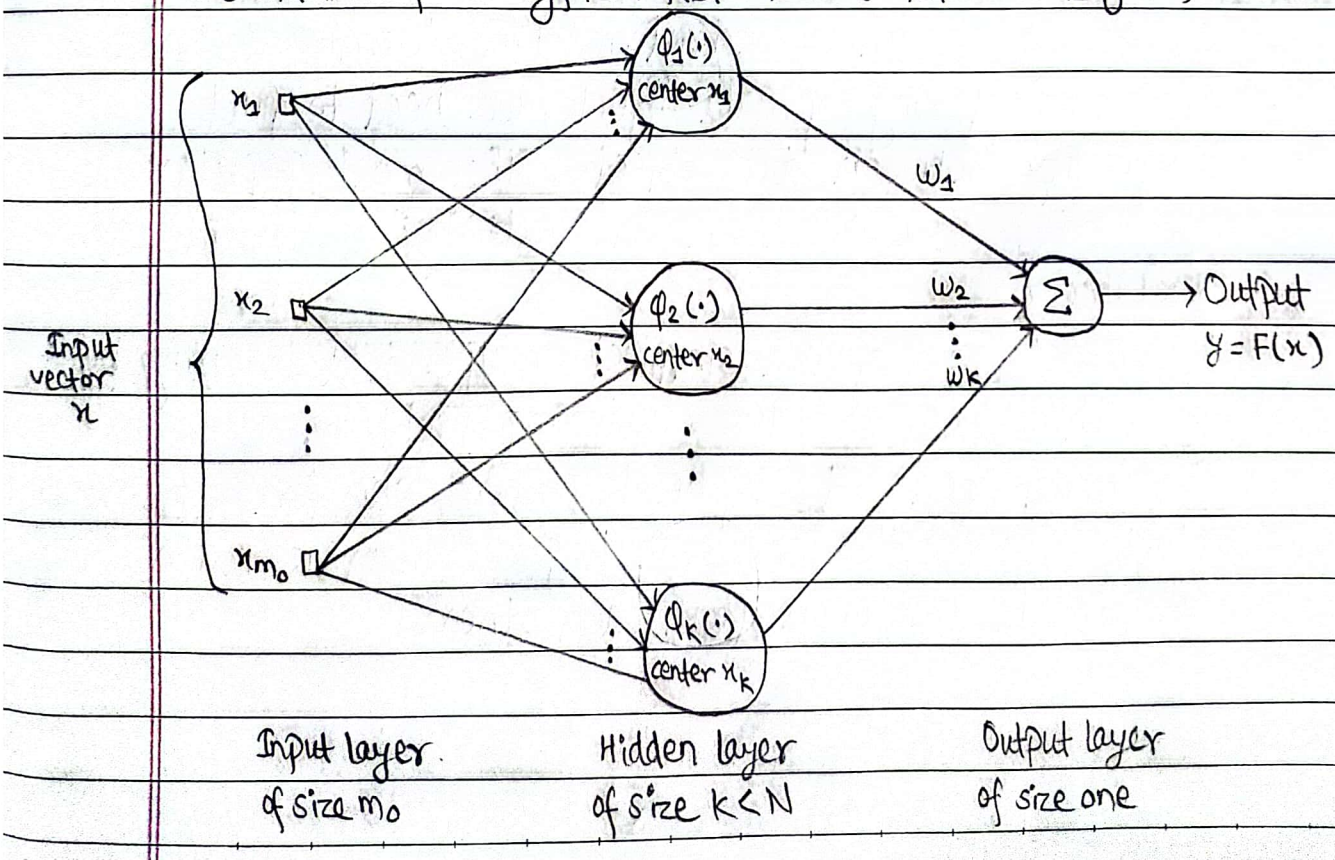
- Input Layer: This layer consists of m_0 source nodes, where m_0 is the dimensionality of the input vector x.

- Hidden Layer: This layer consists of the k computation units ($k \leq N$), where N is number training samples. Each hidden unit is equipped with a radial basis activation function. The RBF is given below.

$$\varphi(x) = \varphi(\|x - x_j\|), \quad j = 1, 2, \ldots k$$

- Output Layer: This layer of neural network contains one or more nodes depending upon the need.

Structure of a typical RBF neural network is given below.



- RBF networks are typically trained from pairs of input and target values by a two-step algorithm.
- In the first step, the center vectors of the RBF functions in the hidden layer are chosen. This step can be performed in several ways; centers can be randomly sampled from some set of examples, or they can be determined using K-means clustering.
- The second step simply fits a model with coefficients to the hidden layer's outputs with respect to some objective function.

K-means Clustering

- The process of grouping a set of data objects into classes of similar objects is called clustering.
- A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters.
- Clustering is unsupervised machine learning technique.
- During clustering of data objects, we need to find similarity between data objects. The most popular similarity measure used in clustering is Euclidean distance.
- K-means is one of the simplest unsupervised learning algorithms used to solve clustering problem.
- The main idea is to define k centers, one for each cluster. The better choice is to place them as much as possible far away from each other.

K-means Algorithm

1. Randomly select K cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center that is closest to the data point.
4. Recalculate the new cluster center using: $C_i = \frac{1}{|C_i|} \sum_{i=1}^{|C_i|} x_i$
5. If no data point change cluster then stop, otherwise repeat go to step 2.

Example:

Divide the data points $\{(1,1), (2,1), (4,3), (5,4)\}$ into two clusters.

Sol:

Let $P1 = (1,1)$ $P2 = (2,1)$ $P3 = (4,3)$ $P4 = (5,4)$

Initial step

Let $C1 = (1,1)$ and $C2 = (2,1)$ are two initial cluster centers.

Iteration 1

Calculate distance between clusters centers and each data points.

$d(C1, P1) = 0$ $d(C2, P1) = \sqrt{(2-1)^2 + (1-1)^2} = 1$

$d(C1, P2) = \sqrt{(2-1)^2 + (1-1)^2} = 1$ $d(C2, P2) = \sqrt{(2-2)^2 + (1-1)^2} = 0$

$d(C1, P3) = \sqrt{(4-1)^2 + (3-1)^2} = 3.6$ $d(C2, P3) = \sqrt{(4-2)^2 + (3-1)^2} = 2.83$

$d(C1, P4) = \sqrt{(5-1)^2 + (4-1)^2} = 5$ $d(C2, P4) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$

Thus after first iteration

Cluster 1 = $\{P1\}$ Cluster 2 = $\{P2, P3, P4\}$

Iteration 2

New cluster centers are : $C1 = (1,1)$ and $C2 = (11/3, 8/3)$

Calculate distance between new cluster centers and each data points.

$d(C1, P1) = 0$ $\quad$ $d(C2, P1) = \sqrt{(11/3-1)^2 + (8/3-1)^2} = 3.1$

$d(C1, P2) = \sqrt{(2-1)^2 + (1-1)^2} = 1$ $\quad$ $d(C2, P2) = \sqrt{(11/3-2)^2 + (8/3-1)^2} = 2.5$

$d(C1, P3) = \sqrt{(4-1)^2 + (3-1)^2} = 3.6$ $\quad$ $d(C2, P3) = \sqrt{(11/3-4)^2 + (8/3-3)^2} = 0.4$

$d(C1, P4) = \sqrt{(5-1)^2 + (4-1)^2} = 5$ $\quad$ $d(C2, P4) = \sqrt{(11/3-5)^2 + (8/3-4)^2} = 1.8$

- Thus after second iteration :

  Cluster 1 = $\{P1, P2\}$ $\quad$ Cluster 2 = $\{P3, P4\}$

- Repeat this process till no re-assignment of data points to clusters.

Iteration 3

New cluster centers are : $C1 = \left(\frac{P_1+P_2}{2}\right) = (3/2, 1)$ and $C2 = \left(\frac{P_3+P_4}{2}\right) = (9/2, 7/2)$

Calculate distance between new cluster centers and each data points.

$d(C1, P1) = \sqrt{(3/2-1)^2 + (1-1)^2} = 0.5$ $\quad$ $d(C2, P1) = \sqrt{(9/2-1)^2 + (7/2-1)^2} = 4.$

$d(C1, P2) = \sqrt{(3/2-2)^2 + (1-1)^2} = 0.5$ $\quad$ $d(C2, P2) = \sqrt{(9/2-2)^2 + (7/2-1)^2} = 3.$

$d(C1, P3) = \sqrt{(4-3/2)^2 + (3-1)^2} = 3.20$ $\quad$ $d(C2, P3) = \sqrt{(9/2-4)^2 + (7/2-3)^2} = 0.7$

$d(C1, P4) = \sqrt{(5-3/2)^2 + (4-1)^2} = 4.60$ $\quad$ $d(C2, P4) = \sqrt{(9/2-5)^2 + (7/2-4)^2} = 0.70$

- Thus after third iteration :

  Cluster 1 = $\{P_1, P_2\}$ $\quad$ Cluster 2 = $\{P_3, P_4\}$

Training RBF Hidden Layer Using K-means

1. Initialization : Choose the cluster centers randomly.
2. Sampling : Draw a sample vector (x) from training set.
3. Similarity Matching : Find Euclidean distance of the sample vector from each cluster center and determine closest (winning) cluster center.

   Mathematically, winning cluster center is given by:

   $k(x) = \arg\min_k(\|x - \mu_k\|)$,

   where, μ_k is cluster center of kth cluster

4. Update Cluster Center : Update winning cluster center as below:

   $$\mu_k = \mu_k + \alpha(x - \mu_k)$$

- After deciding RBF centers, next task is to decide value of parameter sigma used in RBF.
- We can use following equation to determine value of sigma.

  $$\sigma = \frac{d_{max}}{\sqrt{2K}}$$

  , where K is the number of centers and d_{max} is the maximum distance between them.

- Alternatively, we can use following equation to determine the value of sigma.

  $$\sigma = \sqrt{\frac{1}{P}\sum_{k=1}^{P}(x - x_k)^2}$$

  , where P is the number of cluster centers.

Example-

- Consider the XOR problem. Define two RBF centers and calculate the value of sigma for the RBF.

Sol^n:

XOR function is given below.

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

Let us consider (0,0) and (1,1) are two RBF centers and $\varphi_1(x)$ and $\varphi_2(x)$ are two RBFs.

We know that,

$$\sigma = \frac{d_{max}}{\sqrt{2K}}$$

$$\Rightarrow \sigma_1 = \frac{d_{max}}{\sqrt{2K}} = \frac{\sqrt{2}}{\sqrt{4}} = \frac{1}{\sqrt{2}} \quad \text{and,}$$

$$\sigma_2 = \frac{d_{max}}{\sqrt{2K}} = \frac{\sqrt{2}}{\sqrt{4}} = \frac{1}{\sqrt{2}}$$

Thus,

$$\varphi_1(x) = \exp\left(-\frac{\|x-C_1\|^2}{2\sigma^2}\right) = \exp\left(-\|x-C_1\|^2\right)$$

$$\varphi_2(x) = \exp\left(-\frac{\|x-C_2\|^2}{2\sigma^2}\right) = \exp\left(-\|x-C_2\|^2\right)$$

Alternatively,

We know that $\sigma = \sqrt{\frac{1}{P}\sum_{k=1}^{P}(x-x_k)^2}$

Let us choose $P = 2$, thus we get

$$\sigma_1 = \sqrt{\frac{1}{2}\{(0-1)^2 + (0-0)^2\} + \{(0-0)^2 + (0-1)^2\}} = 1$$

$$\sigma_2 = \sqrt{\frac{1}{2}\{(1-1)^2 + (1-0)^2 + \{(1-0)^2 + (1-1)^2\}} = 1$$

Thus,

$$\varphi_1(x) = \exp\left(-\frac{\|x - C_1\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|x - C_1\|^2}{2}\right)$$

$$\varphi_2(x) = \exp\left(-\frac{\|x - C_2\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|x - C_2\|^2}{2}\right)$$

LMS Estimation of Weight Vector.

- Let us assume that N is the number of input samples in the training set, M is the dimension of hidden layer and C is the number of classes (i.e. number of output nodes)
- For notational simplicity assume that $\varphi_j(x_i) = \varphi_{ji}$
- For the properly adjusted weights, following relation must hold

$$\begin{bmatrix} \varphi_{11} & \varphi_{21} & \cdots & \varphi_{M1} \\ \varphi_{12} & \varphi_{22} & \cdots & \varphi_{M2} \\ \vdots & \vdots & & \\ \varphi_{1N} & \varphi_{2N} & \cdots & \varphi_{MN} \end{bmatrix} \begin{bmatrix} w_{k1} \\ w_{k2} \\ \vdots \\ w_{kM} \end{bmatrix} = \begin{bmatrix} d_{k1} \\ d_{k2} \\ \vdots \\ d_{kN} \end{bmatrix}$$

where, d_{kj} is the desired value of summing function of kth output neuron for j^{th} input

- We know that

$$d_{kj} = 1, \quad \text{if } x_j \in H_k \text{ and}$$
$$d_{kj} = -1, \quad \text{if } x_j \notin H_k$$

- Above equation can be written ~~as~~ in simplified form as below:

$$\varphi_{ji} \cdot w_{kj} = d_{ki}, \quad i = 1 \ldots N, \; j = 1 \ldots M, \; k = 1 \ldots C$$

- However, if the network is not properly trained, above equation is not satisfied. In this case the error term is given as below.

$$e = \varphi w_k - d_k$$

- Now, the energy function can be written as

$$E(w_k) = \frac{1}{2} \sum e^2 = \frac{1}{2} \sum (\varphi w_k - d_k)^2$$

- Now, ~~the~~ Using gradient descent algorithm

$$\nabla E = \varphi^T (\varphi w_k - d_k)$$

- Equating above equation with zero and rearranging the terms, we get

$$w_k = (\varphi^T \varphi)^{-1} \varphi^T d_k$$

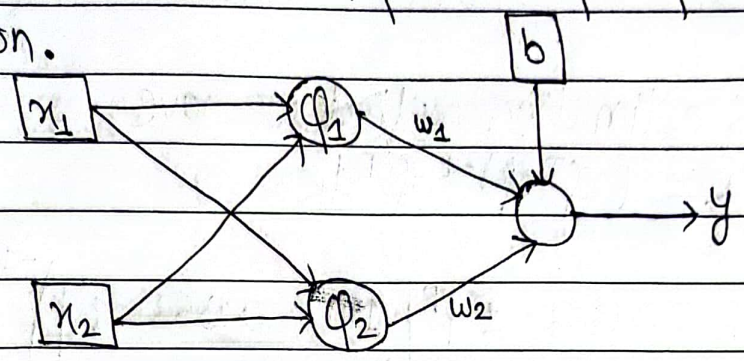
Date
Page

Example:

- Devise RBF neural network for XOR function using two cluster centers.

Sol^n:

Since we are assuming two cluster centers, structure of RBF network for XOR function will be like below.
Assume that Activation function of output layer is step function.



- Assume that (0,0) and (1,1) are two RBF centers. Thus, we have following values of sigma for two RBFs.

$$\sigma = \frac{d_{max}}{\sqrt{2K}}$$

$$\Rightarrow \sigma_1 = \frac{d_{max}}{\sqrt{2K}} = \frac{\sqrt{2}}{\sqrt{4}} = \frac{1}{\sqrt{2}} \quad \text{and}$$

$$\sigma_2 = \frac{d_{max}}{\sqrt{2K}} = \frac{\sqrt{2}}{\sqrt{4}} = \frac{1}{\sqrt{2}}$$

- Thus we have,

$$\varphi_1(x) = \exp\left(-\frac{\|x-c_1\|^2}{2\sigma^2}\right) = \exp\left(-\|x-c_1\|^2\right)$$

$$\varphi_2(x) = \exp\left(-\frac{\|x-c_2\|^2}{2\sigma^2}\right) = \exp\left(-\|x-c_2\|^2\right)$$

- Now, we get following output from hidden layer of RBF network.

x_1	x_2	φ_1	φ_2
0	0	1.0	0.1353
0	1	0.3678	0.3678
1	0	0.3678	0.3678
1	1	0.1353	1.0

- From LMS estimation method we have

$$w_k = (\varphi^T \varphi)^{-1} \varphi^T d_k$$

$$w = \left(\begin{bmatrix} 1.0 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1.0 & 1 \end{bmatrix}^T \begin{bmatrix} 1.0 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1.0 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1.0 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1.0 & 1 \end{bmatrix}^T \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

$$= \begin{array}{c|c} \begin{bmatrix} -5.02 \\ -5.02 \\ 4.68 \end{bmatrix} & \begin{array}{c} w_1 \\ w_2 \\ b \end{array} \end{array}$$

Learning Procedure for RBF Networks

- Learning in RBF network use K-means and LMS of RLS algorithm. This hybrid learning procedure employed in RBF networks is summarized below.
- Size of input layer is determined by dimensionality of training sample, let m_0 is the dimensionality of training sample.
- Size of input layer is determined by number of clusters, let $K = m_1$ is the number of clusters.
- Use K-means algorithm to determine cluster centers.
- Determine value of parameter σ (RBF width) using following formula.

  $\sigma = \frac{d_{max}}{\sqrt{2K}}$, where K is the number of centers and d_{max} is the maximum distance between them.

- Determine the values of parameters (weights and bias) using LMS.

RBFNN for Function Approximation and Classification

- In case of function approximation output layer of RBFNN have only one node.

- Whereas in case of classification number of nodes in output layer of RBFNN depends upon the classification problem we are trying to solve.

- For classification, you typically have one node per output category, each outputting a score for their respective category.

Bias Variance Tradeoff.

- The prediction error for any machine learning algorithm can be broken down into three parts:
  - Bias Error
  - Variance Error
  - Irreducible Error

- The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

- Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.

- Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

- Bias is the tendency of an estimator to pick a model for the data that is not structurally correct. For example, suppose that we use a linear regression model on a cubic function.

- Bias is caused by underfitting and is the result of simplifying assumptions made by a model to make the target function easier to learn.

- Variance is taken as the variability of a model prediction for a given data point.

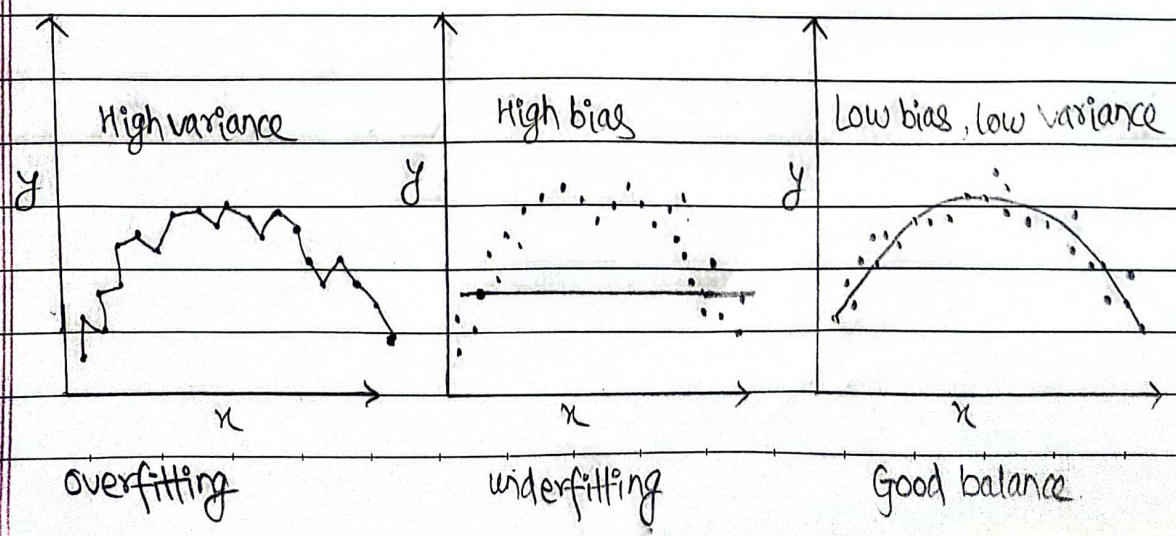
- Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn't seen before. As a result, such models perform very well on training data but has high error rates on test data.

- Variance is the result of overfitting.

- Variance is error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs.

- In supervised learning, underfitting happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance.

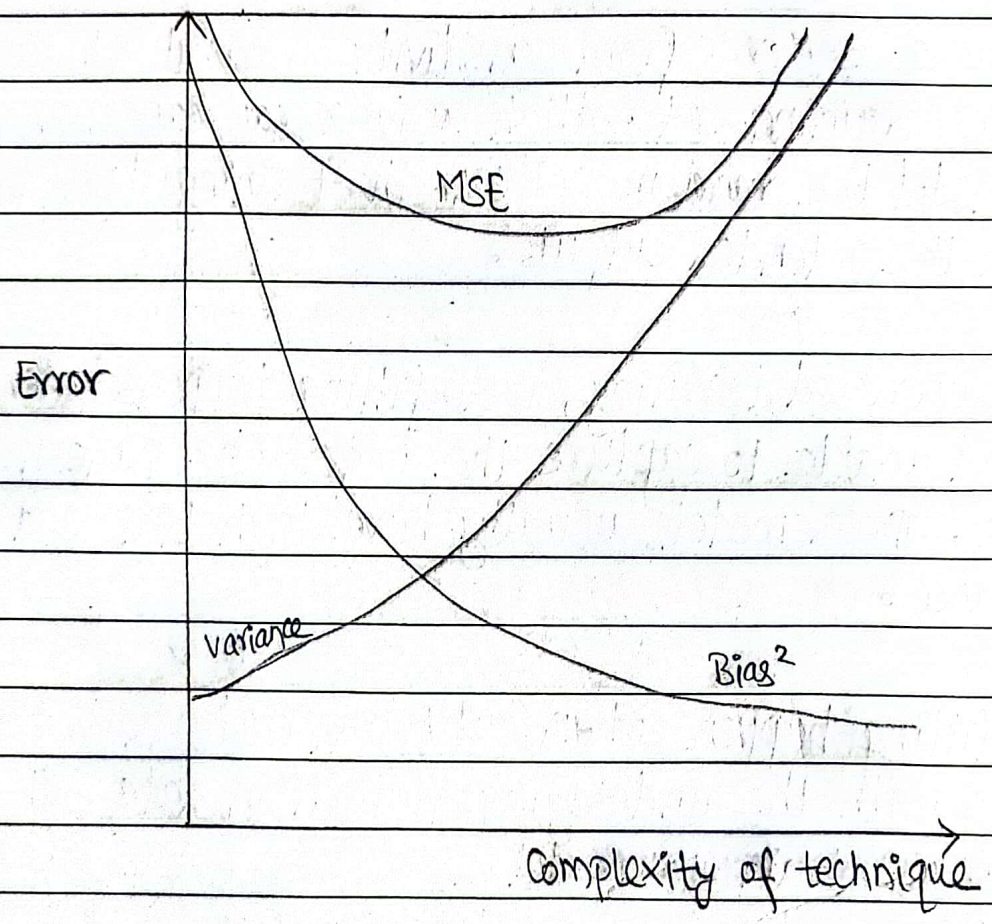
- Overfitting happens when our model captures the noise along with the underlying pattern in data. It happens when we train our model a lot over noisy dataset. These models have low bias and high variance.



- If our model is too simple and has very few parameters then it may have high bias and low variance.

- On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the good balance without overfitting and underfitting the data. This keeps the MSE minimal.

$$MSE = Bias^2 + Variance + Irreducible\ Error$$



Regularization.

- Regularization is a technique used in an attempt to solve the overfitting problem.
- A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new input test data.
- Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error.
- In another way, we can say that any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error is regularization.
- There are many regularization strategies. Some put extra constraints on a machine learning model, such as adding restrictions on the parameter values. Some add extra terms in the objective function. These strategies are collectively known as regularization.
- A network with large network weights can be a sign of an unstable network where small changes in the input can lead to large changes in the output. This can be a sign that the network has overfit the training dataset and will likely perform poorly when making predictions on new data.
- A solution to this problem is to update the learning algorithm to encourage the network to keep the weights small. This is called weight regularization and it can be used as a general technique to reduce overfitting of the training dataset and improve the generalization of the model.

How to Penalize Large Weights

- There are two parts to penalizing the model based on the size of the weights.
- The first is the calculation of the size of the weights, and the second is the amount of attention that the optimization process should pay to the penalty.
- Neural network weights are real-values that can be positive or negative, as such, simply adding the weights is not sufficient.

- There are two main approaches used to calculate the size of the weights, they are:
  - Calculate the sum of the absolute values of the weights, called L1.
  - Calculate the sum of the squared values of the weights, called L2.

L1 Regularization

- In this approach, we penalize the absolute value of the weights as below:

$$L(x) = L(x) + \lambda \sum |w|$$

- L1 encourages weights to 0.0 if possible, resulting in more sparse weights (weights with more 0.0 values).

- The sparsity property induced by L1 regularization has been used extensively as a feature selection mechanism.

L2 Regularization

- In this approach, we penalize the absolute value of the weights as below:

$$L(x) = L(x) + \lambda \sum w^2$$

- L2 offers more nuance, both penalizing larger weights more severely, but resulting in less sparse weights.

- L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

Control Impact of the Penalty

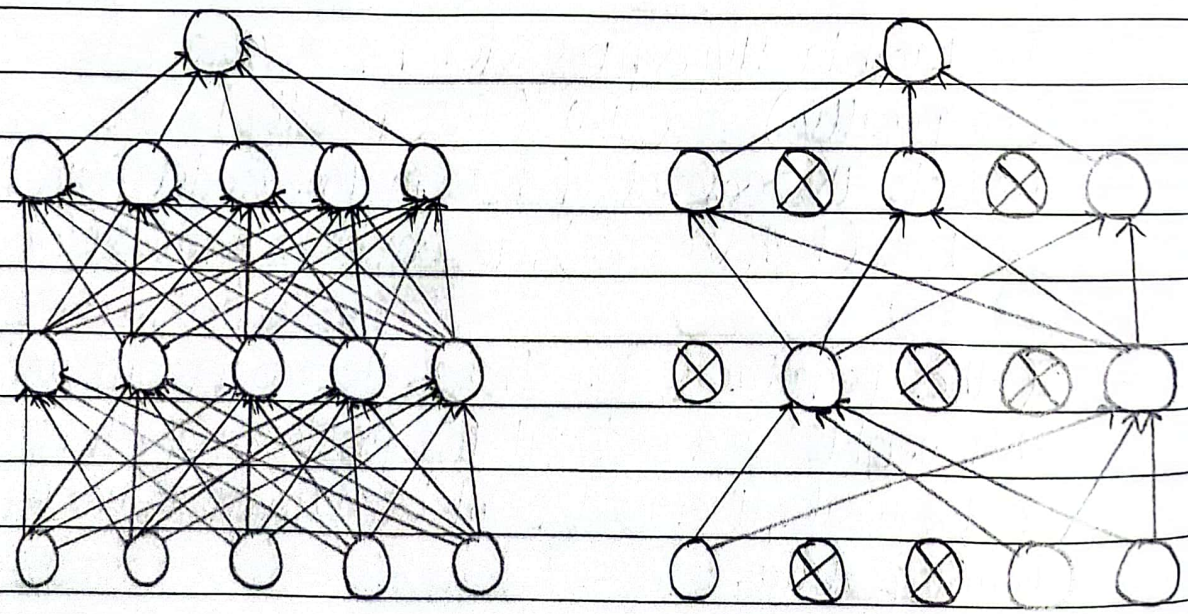
- The calculated size of the weights is added to the loss objective function when training the network.
- Rather than adding each weight to the penalty directly, they can be weighted using a new Hyperparameter called lambda. This controls the amount of attention that the learning process should pay to the penalty.

- The Lambda Hyperparameter has a value between 0.0 (no penalty) and 1.0 (full penalty). This Hyperparameter controls the amount of bias in the model from 0.0, or low bias (high variance), to 1.0, or high bias (low variance).

- If the penalty is too strong, the model will underestimate the weights and underfit the problem. If the penalty is too weak, the model will be allowed to overfit the training data.

Dropout Regularization

- The term "dropout" refers to dropping out units (both hidden and visible) in a neural network.
- Simply put, dropout refers to ignoring certain set of units / neurons, which is chosen at random, during the training phase. These units are not considered during a particular forward or backward pass.
- Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
- A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.
- Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.



(a) Standard Neural Net (b) After applying dropout

Data Augmentation

- The simplest way to reduce overfitting is to increase the size of the training data. We were not able to increase the size of training data always.
- In case of digit identification, there are a few ways of increasing the size of the training data - rotating the image, flipping, scaling, shifting, etc.
- This technique is known as data augmentation. This usually provides a big leap in improving the accuracy of the model. It can be considered as a mandatory trick in order to improve our predictions.

Unit 5 :
Kernel Methods for Pattern Analysis

Statistical Learning Theory

- Statistical Learning theory is a framework for machine learning that draws from statistics and functional analysis.
- It deals with finding a predictive function based on the data presented.
- The main idea in statistical learning theory is to build a model that can draw conclusions from data and make predictions.
- With statistical learning theory, there are two ~~many~~ main types of data:
  - Dependent variable : A variable (y) whose values depend on the values of other variables
    (a dependent variable is sometimes also referred to as a target variable).

  - Independent variables: A variable (x) whose value does not depend on the values of other variables.
    (independent variables are sometimes also referred to as predictor variables, input variables, explanatory variables, or features)

- A statistical model defines the relationships between a dependent and independent variable.
- For example, if the relationships between the size of the home (x) and the price of the home (y) is linear. We can define this relationship by using $y = ax + b$ where 'a' represents the gradient and 'b' is the intercept.

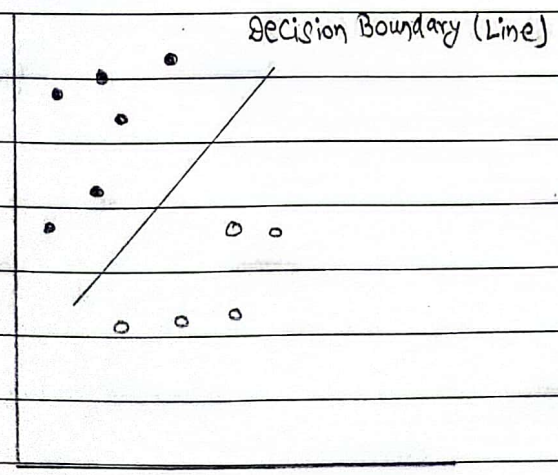
If we suppose that the price (y) of the depends upon size (x_1) of the home and the number of bathrooms (x_2). We can define this relationship by using $y = ax_1 + bx_2 + c$.

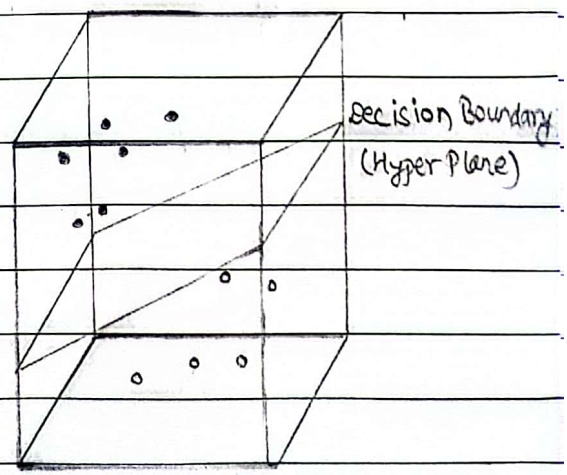
Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

- A support vector machine takes input data points and outputs the hyperplane (which in two dimensions it's simply a line) that best separates the data points into two classes.

- This line or hyperplane is the decision boundary : any data points that falls to one side of it is classified in one class and, and the data points that falls to the other of it is ~~cat~~ classified in another class.





- Compared to newer algorithms like neural networks, they have two main advantages: higher speed and better performance with a limited number of samples (in the thousands).

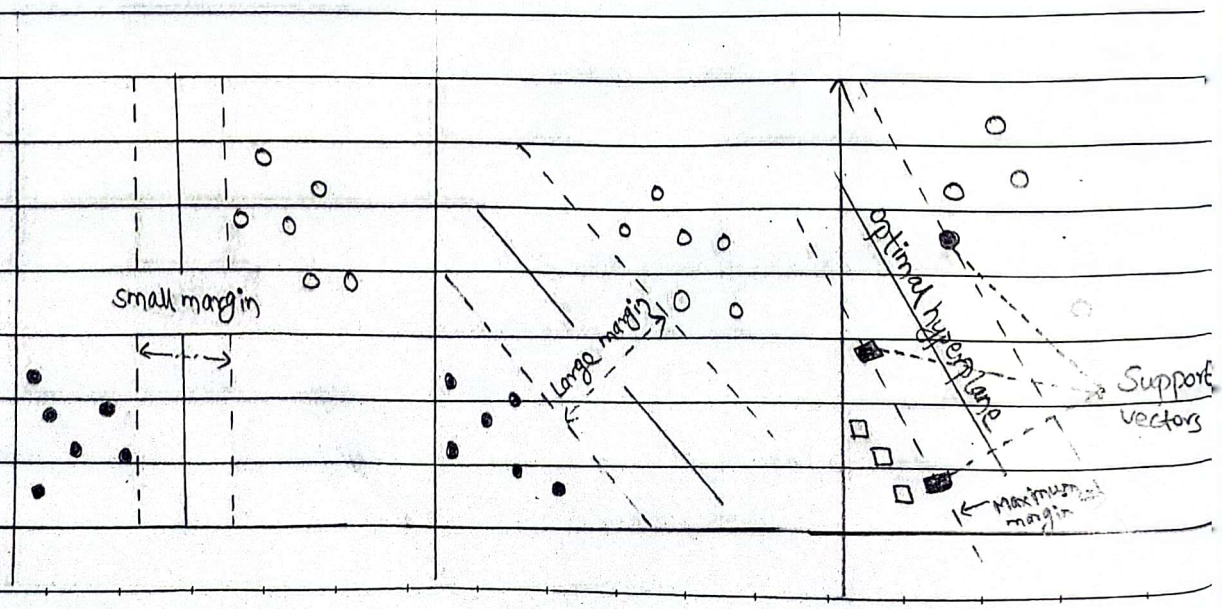
- Support vectors are data points that are closest to the hyperplane and influence the position and orientation of the hyperplane.

- To separate the two classes of data points, there are many possible hyperplanes that could be chosen.

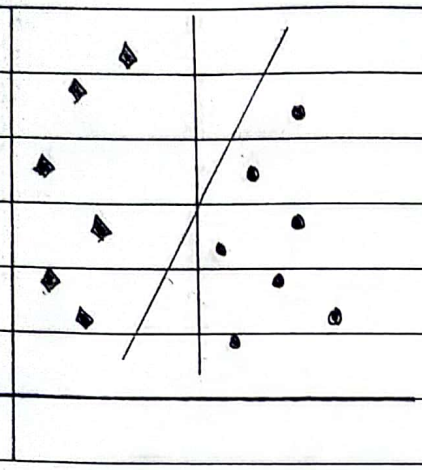
- SVM algorithm selects optimal hyperplane by choosing hyperplane with largest margin. Such hyperplane is called maximum marginal hyperplane (MMH).

- Let H1 and H2 are planes that passes through support vectors and parallel to the hyperplane of decision boundary.

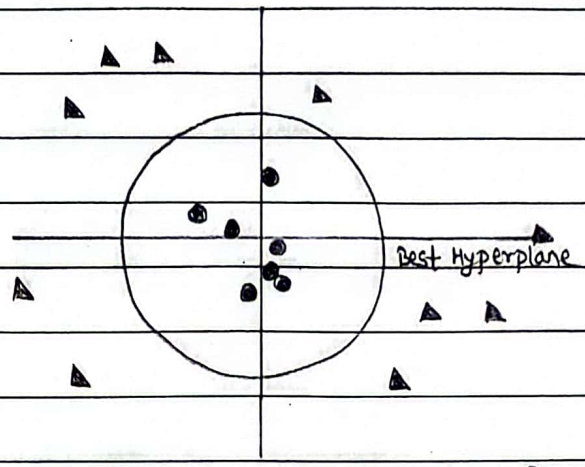
- Distance between plane H1 and the hyperplane should be equal to distance between plane H2 and the hyperplane. Margin is the distance between planes H1 and H2.



- Support Vector Machine (SVM) can be of two types: Linear SVM and Non-linear SVM.
- SVM that is used to classify linearly separable data points is called linear SVM whereas the SVM that is used to classify non-linearly separable data points is called non-linear SVM.



Linearly separable Data Points



Non-linearly separable Data Points.

- Non-linear SVM works in following two steps:
  - It transforms low-dimensional data points into high-dimensional data points, that are linearly separable, by using kernel-trick.
  - Then, it classifies data points using linear-hyperplane.

- SVM algorithms use a set of mathematical functions that are defined as the kernel.

- These functions are used to transform non-linearly separable low-dimensional data points into linearly separable high-dimensional data points.

- Popular kernels are: Linear Kernel, Polynomial Kernel, Gaussian Kernel, Radial Basis Function (RBF), Sigmoid Kernel, etc.

- Consider the following 2-D data points which are linearly inseparable. We can transform the data points into linearly separable data points by adding third dimension $z = x^2 + y^2$.

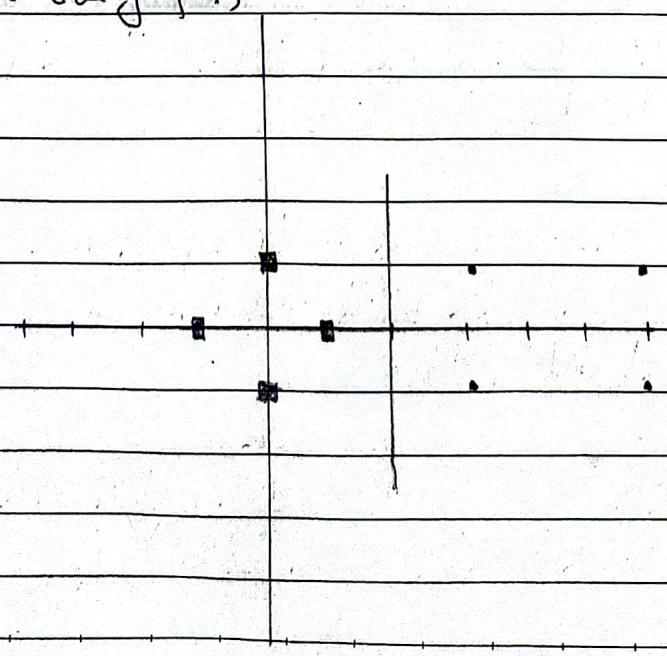
Example:

Consider the following data points:

- Positively Labelled Data Points: $(3,1), (3,-1), (6,1), (6,-1)$
- Negatively Labelled Data Points: $(1,0), (0,1), (0,-1), (-1,0)$.

Determine the equation of hyperplane that divides the above data points into two classes:

Let's draw the graph,



Now, from above graph, there are 3 support vectors, they are :

$S1 = (1,0)$, $S2 = (3,1)$, $S3 = (3,-1)$

Augment support vectors with $b=1$

$S1 = (1,0,1)$, $S2 = (3,1,1)$, $S3 = (3,-1,1)$

Since, there are three support vectors, we need to calculate three variables.

Thus, three linear equations can be written as :

$$\alpha S1 \cdot S1 + \beta S2 \cdot S1 + \lambda S3 \cdot S1 = -1$$
$$\alpha S1 \cdot S2 + \beta S2 \cdot S2 + \lambda S3 \cdot S2 = 1$$
$$\alpha S1 \cdot S3 + \beta S2 \cdot S3 + \lambda S3 \cdot S3 = 1$$

~~After~~ simplifying above equations; ~~we get~~

$$\alpha (1,0,1)\cdot(1,0,1) + \beta (3,1,1)\cdot(1,0,1) + \lambda (3,-1,1)\cdot(1,0,1)$$
$$= \alpha(1+0+1) + \beta(3+0+1) + \lambda(3+0+1) = 2\alpha + 4\beta + 4\lambda.$$

$$\alpha (1,0,1)\cdot(3,1,1) + \beta (3,1,1)\cdot(3,1,1) + \lambda (3,-1,1)\cdot(3,1,1)$$
$$= \alpha(3+0+1) + \beta(9+1+1) + \lambda(9-1+1) = 4\alpha + 11\beta + 9\lambda$$

$$\alpha (1,0,1)\cdot(3,-1,1) + \beta (3,1,1)\cdot(3,-1,1) + \lambda (3,-1,1)\cdot(3,-1,1)$$
$$= \alpha(3+0+1) + \beta(9-1+1) + \lambda(9+1+1) = 4\alpha + 9\beta + 11\lambda$$

After simplifying above equations, we get

$$2\alpha + 4\beta + 4\lambda = -1 \quad \text{—— (I)}$$
$$4\alpha + 11\beta + 9\lambda = 1 \quad \text{—— (II)}$$
$$4\alpha + 9\beta + 11\lambda = 1 \quad \text{—— (III)}$$

Solving eqⁿ (I) and (III) and subtracting (III) from (II)

$$\cancel{4\alpha} + 11\beta + 9\lambda = 1$$
$$\cancel{4\alpha} + 9\beta + 11\lambda = 1$$
$$- \quad - \quad - \quad -$$

$$2\beta - 2\lambda = 0$$
$$2\beta = 2\lambda$$
$$\beta = \lambda \quad \text{— (IV)}$$

Multiplying eqⁿ (I) by 2 and subtracting from eqⁿ (II) we get;

$$\cancel{4\alpha} + 8\beta + 8\lambda = -2$$
$$\cancel{4\alpha} + 11\beta + 9\lambda = 1$$
$$- \quad - \quad - \quad -$$

$$-3\beta - 1\lambda = -3$$
$$-\lambda = -3 + 3\beta$$
$$\lambda = 3 - 3\beta \quad \text{— (V)}$$

Putting value of β in eqⁿ (V)

$$\lambda = 3 - 3\lambda$$
$$4\lambda = 3$$
$$\lambda = \frac{3}{4} = 0.75$$

Since from eqⁿ (IV)

$$\lambda = \beta$$
$$\beta = 0.75$$

Putting values of λ and β to eqⁿ (I), we get;

$$2\alpha + 4 * 0.75 + 4 * 0.75 = -1$$
$$2\alpha + 3 + 3 = -1$$
$$2\alpha = -7$$
$$\alpha = -3.5$$

We get $\alpha = -3.5$, $\beta = 0.75$, $\lambda = 0.75$

Now, we can compute weight vector of hyperplane as below:

$$w = \alpha S_1 + \beta S_2 + \lambda S_3$$
$$= -3.5 * (1,0,1) + 0.75 * (3,1,1) + 0.75 * (3,-1,1)$$
$$= (-3.5, 0, -3.5) + (2.25, 0.75, 0.75) + (2.25, -0.75, 0.75)$$
$$= (1, 0, -2)$$

Thus, $w = (1, 0)$ and $b = -2$.

Thus, eqn of hyperplane is:

$$WX + b = 0$$
$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} (x, y) + (-2) = 0$$
$$x - 2 = 0$$

Support Vector Regression:

- Support Vector Regression (SVR) uses the same principle as SVM, but for regression problems.

- The problem of regression is to find a function that approximates mapping from an input domain to real numbers on the basis of a training sample.

- In Support Vector Regression, the straight line that is required to fit the data is referred to as hyperplane.

- Key parameters that are used in Support Vector Regression are discussed below:
  - Hyperplane:

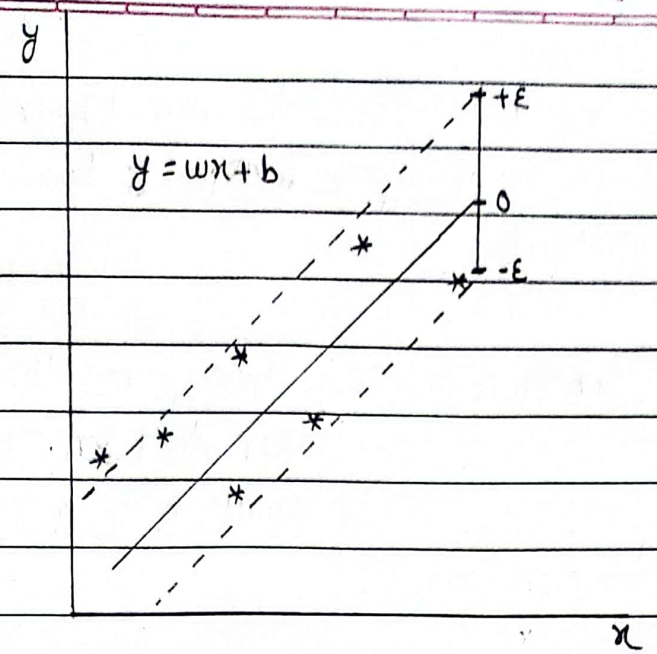
    Hyperplanes are decision boundaries that is used to predict the continuous output. The data points on either side of the hyperplane that are closest to the hyperplane are called Support Vectors. These are used to plot the required line that shows the predicted output of the algorithm.

  - Kernel:

    A kernel is a set of mathematical functions that takes data as input and transform it into the required form. These are generally used for finding a hyperplane in the higher dimensional space.

  - Boundary Lines:

    These are the two lines that are drawn around the hyperplane at a distance of ε (epsilon). It is used to create a margin between the data points.



- The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points.

- Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value.

- The threshold value is the distance between the hyperplane and boundary line.

- The fit time complexity of SVR is more than quadratic with the number of samples which makes it hard to scale to datasets with more than a couple of 10000 samples.

Kernel Functions :

- Kernels are a way to solve non-linear problems with the help of linear classifiers. This is known as the kernel trick method.

- Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transform to a linear equation in a higher number of dimension spaces.

- Let us see some of the kernel function or the types that are being used in SVM :

- Linear kernel :

  ~~lin~~ Let us say that we have two vectors with name $x1$ and $y1$, then the linear kernel is defined by the dot product of these two vectors :

  $$K(x1, x2) = x1 \cdot x2$$

- Polynomial kernel :

  A polynomial kernel is defined by the following equation :

  $$K(x1, x2) = (x1 \cdot x2 + 1)^d$$

  where, d is the degree of the polynomial and $x1$ and $x2$ are vectors.

- Gaussian kernel :

  This kernel is an example of a radial basis function kernel. Below is the equation for this.

  $$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

- Exponential kernel:

This is in close relation with the previous kernel i.e. the Gaussian kernel with the only difference is — the square of the norm is removed. The function of the exponential function is:

$$k(x,y) = \exp\left(\frac{-\|x-y\|}{2\sigma^2}\right)$$

- Laplacian kernel:

This type of kernel is less prone for changes and is totally equal to previously discussed exponential function kernel, the equation of Laplacian kernel is given as:

$$k(x,y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right).$$

UNIT 6:
Self-Organizing Maps

Introduction of SOM

- A self-organizing map (SOM) is a type of ANN that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional) discretized representation of the input space of the training samples.
- This low dimensional representation can be viewed as a map. Therefore SOM is a method to do dimensionality reduction.
- Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning.
- In SOM all inputs are fully connected with the output neurons and those neurons competes with each other. An output neuron that win the competition is called winning neuron or winner takes all neuron.
- Synaptic weights are adjusted in the favor of winning neuron so that when same or similar input pattern is presented to the neuron there will be high chance of winning the competition for the neuron.
- This means weights of winning neurons are updated such that Euclidean distance between the input and weights of the neuron is minimized.
- In SOM, output neurons are organized in the form of one or two dimensional lattices. Higher dimensional lattices of neurons are also possible but are not practically common.

- SOMs are commonly used as visualization aids. They can make it easy for us humans to see relationships between vast amounts of data.
- Example: World Poverty Map
- A SOM has been used to classify statistical data describing various quality-of-life factors such as state of health, nutrition, educational services, etc. Countries with similar quality-of-life factors end up clustered together.

- The main goal of the SOM is to transform an incoming pattern of arbitrary dimension into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion.
- Each output neuron is fully connected to all the source nodes in the input layer.
- This network represents a feedforward structure with a single computational layer consisting of neurons arranged in a 2D or 1D grid.

- The algorithm which is responsible for the self-organization of the network is based on three essential steps:

- Competition: For each input pattern, the neurons in the network compute their respective values of a discriminant. The neuron with the largest value of discriminant function is declared winner of the competition.

- Cooperation: The winning neuron determines the spatial location of a topological neighborhood of excited neurons. This is definitely cooperation because exited neuron not only strengthen itself but also it strengthens neighboring neurons.

- Synaptic Adaptation: This mechanism enables the excited neurons to increase their individual values of the discriminant function in relation to the input pattern through suitable adjustments applied to their synaptic weights.

- Let us examine mathematical modeling of each of above mechanism used in self organizing maps.

Competition

- Let m be the dimension of the input space. A pattern chosen randomly from input space is denoted by:

$$x = [x_1, x_2, \ldots, x_m]^T$$

- The synaptic weight of each neuron in the output layer has the same dimension as the input space. We denote the weight of neuron j as:

$$w_j = [w_{j1}, w_{j2}, \ldots, w_{jm}]^T, \quad j = 1, 2, \ldots, L$$

where, L is the total number of neurons in the output layer.

- To find the best match of the input vector x with the synaptic weights w_j we use the Euclidean distance. The neuron with the smallest distance is called $i(x)$ and is given by:

$$i(x) = \arg\min_j \| x - w_j \|, \quad j = 1, 2, \ldots L.$$

- The neuron (i) that satisfies the above condition is called best-matching or winning neuron for the input vector x

Cooperation

- The winning neuron effectively locates the center of a topological neighborhood.
- The neighborhood should be a decreasing function of the lateral distance between the neurons.
- Let d_{ij} is the lateral distance between neurons i and j and let us denote h_{ji} as the topological neighborhood around neuron i.
- Thus h_{ji} must be function of distance which satisfies the following two requirements:
  - The topological neighborhood h_{ji} is symmetric about the maximum point defined by $d_{ij} = 0$; in other words, it attains its maximum value at the winning neuron i for which the distance is zero.
  - The amplitude of the topological neighborhood h_{ji} decreases monotonically with increasing lateral distance d_{ij} decaying to zero for $d_{ij} \to \infty$; this is a necessary condition for convergence.
- A typical choice of h_{ji} is the Gaussian function which is given by:

$$h_{ji}(\vec{x}) = \exp\left(-\frac{d_{ij}^2}{2\sigma^2}\right)$$

The parameter σ is the width of the neighborhood.

- Another characteristic feature of the SOM algorithm is that the size of the neighborhood shrinks with time. This requirement is satisfied by making the width of the Gaussian function decreasing with time.

- A popular choice is the exponential decay described by:

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \qquad n = 0, 1, 2, \ldots$$

where σ_0 is the value of σ at the initialization of the SOM algorithm and τ_1 is a time (iteration) constant.

- Corresponding the neighborhood function assumes a time dependent ~~fn~~ form of its own:

$$h_{ji(\vec{x})}(n) = \exp\left(-\frac{d_{ji}^2}{2\sigma(n)^2}\right) \qquad n = 0, 1, 2, \ldots$$

Adaptation

- The adaptive process modifies the weights of the network so as to achieve the self-organization of the network.
- Only the winning neuron and neurons inside its neighborhood have their weights adapted. All the other neurons have no change in their weights.
- A method for deriving the weight update equations for the SOM model is based on a modified form of Hebbian learning. There is a forgetting term in the standard Hebbian weight equations.
- Let us assume that the forgetting term has the form $g(y_j)\, w_j$ where y_j is the response of neuron j and $g(\cdot)$ is a positive scalar function of y_j.
- The only requirement for the function $g(y_j)$ is that the constant term in its Taylor series expansion to be zero when the activity is zero, i.e.: $g(y_j) = 0$ for $y_j = 0$
- The modified Hebbian rule for the weights of the output neurons is given by:

$$\Delta w_j = \alpha y_j x - g(y_j)\, w_j$$

where, α is the learning rate parameter of the algorithm.

Date
Page

- To satisfy the requirement for a zero constant term in the Taylor series we choose the following form for the function $g(y_j)$:

$$g(y_j) = \alpha y_j$$

- We can simplify further by setting:

$$y_j = h_{ji}(x)$$

- Combining the previous equations we get:

$$\Delta w_j = \alpha h_{ji(x)} (x - w_j)$$

- Finally using a discrete representation for time we can write:

$$w_j(n+1) = w_j(n) + \alpha(n) h_{ji(x)}(n) (x - w_j(n))$$

- The learning rate must also be time varying. A suitable form is given by:

$$\alpha(n) = \alpha_0 \exp\left(\frac{-n}{\tau_2}\right) \quad n = 0, 1, 2, \ldots$$

where n_0 is an initial value and τ_2 is another time constant of the SOM algorithm.

Summarized Algorithm

1. Initialization: Choose random values for the initial weight vectors $w_j(0)$. The weight vectors must be different for all neurons. Usually we keep the magnitude of the weights small.

2. Sampling: Draw a sample x from the input space with a certain probability; the vector x represents the activation pattern that is applied to the lattice. The dimension of x is equal to m.

3. Similarity Matching: Find the best-matching (winning) neuron $i(x)$ at time step n by using the minimum Euclidean distance criterion.

$$i(x) = \arg\min_j \|x - w_j\|, \quad j = 1, 2, \ldots, l$$

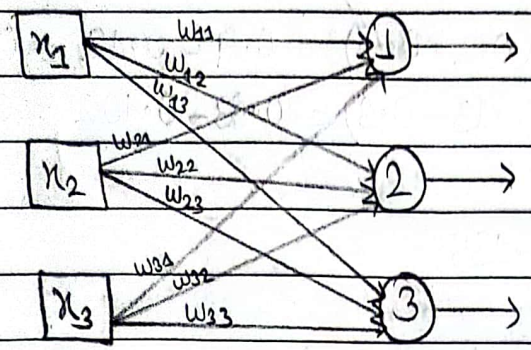
4. Updating: Adjust the synaptic weight vectors of all neurons by using the update formula:

$$w_j(n+1) = w_j(n) + \alpha(n)\, h_{j,i(x)}(n)\, (x(n) - w_j)$$

5. Continuation: Continue with step 2 until no noticeable changes in the feature map are observed.

Example:

Consider following 1-D SOM and 3-D inputs. Show the working of SOM for the given two inputs.



x_1	x_2	x_3
0.1	0.2	0.13
0.5	0.3	0.7

Initial weight matrix

$w_{11} = 0.1$	$w_{12} = 0.2$	$w_{13} = 0.3$
$w_{21} = 0.2$	$w_{22} = 0.4$	$w_{23} = 0.6$
$w_{31} = 0.1$	$w_{32} = 0.3$	$w_{33} = 0.5$

Sol:

Iteration 1: input (0.1, 0.2, 0.13)

Find Euclidean distance between the input and weight vector of each output neuron.

$d_1 = d(x, w_1) = \sqrt{(0.1-0.1)^2 + (0.2-0.2)^2 + (0.13-0.1)^2} = 0.03$

$d_2 = d(x, w_2) = \sqrt{(0.1-0.2)^2 + (0.2-0.4)^2 + (0.13-0.3)^2} = 0.28$

$d_3 = d(x, w_3) = \sqrt{(0.1-0.3)^2 + (0.2-0.6)^2 + (0.13-0.5)^2} = 0.58$

Clearly neuron 1 is winner.

Update weights: Assume $\alpha = 1$ $\sigma = 1$

We know that,

$$w_j(n+1) = w_j(n) + \alpha h_{ji}(x - w_j(n)) \qquad h_{ji} = \exp\left(-\frac{d_{ji}^2}{2\sigma^2}\right)$$

$\Rightarrow$

$h_{11} = 1$

$w_{11} = w_{11} + 1*1*(0.1-0.1) = 0.1$

$w_{21} = w_{21} + 1*1*(0.2-0.2) = 0.2$

$w_{31} = w_{31} + 1*1*(0.13-0.1) = 0.13 + 0.03 = 0.163$

Similarly,

$$h_{21} = \exp\left(\frac{-1}{2}\right) = 0.6$$

$$w_{12} = w_{12} + 1 * 0.6 * (0.1 - 0.2) = 0.2 - 0.06 = 0.14$$

$$w_{22} = w_{22} + 1 * 0.6 * (0.2 - 0.4) = 0.4 - 0.12 = 0.28$$

$$w_{32} = w_{32} + 1 * 0.6 * (0.13 - 0.3) = 0.3 - 0.102 = 0.198$$

$$h_{31} = \exp\left(\frac{-4}{2}\right) = 0.135$$

$$w_{13} = w_{13} + 1 * 0.135 * (0.1 - 0.3) = 0.3 - 0.027 = 0.273$$

$$w_{23} = w_{23} + 1 * 0.135 * (0.2 - 0.6) = 0.6 - 0.054 = 0.546$$

$$w_{33} = w_{33} + 1 * 0.135 * (0.13 - 0.5) = 0.5 - 0.049 = 0.45$$

$w_{11} = 0.1$ $w_{12} = 0.14$ $w_{13} = 0.273$
$w_{21} = 0.2$ $w_{22} = 0.28$ $w_{23} = 0.546$
$w_{31} = 0.13$ $w_{32} = 0.198$ $w_{33} = 0.45$

Iteration 2: input $(0.5, 0.3, 0.7)$

Find Euclidean distance between the input and weight vector of each output neuron.

$$d_1 = d(x, w_1) = \sqrt{(0.5 - 0.1)^2 + (0.3 - 0.2)^2 + (0.7 - 0.13)^2} = 0.7$$

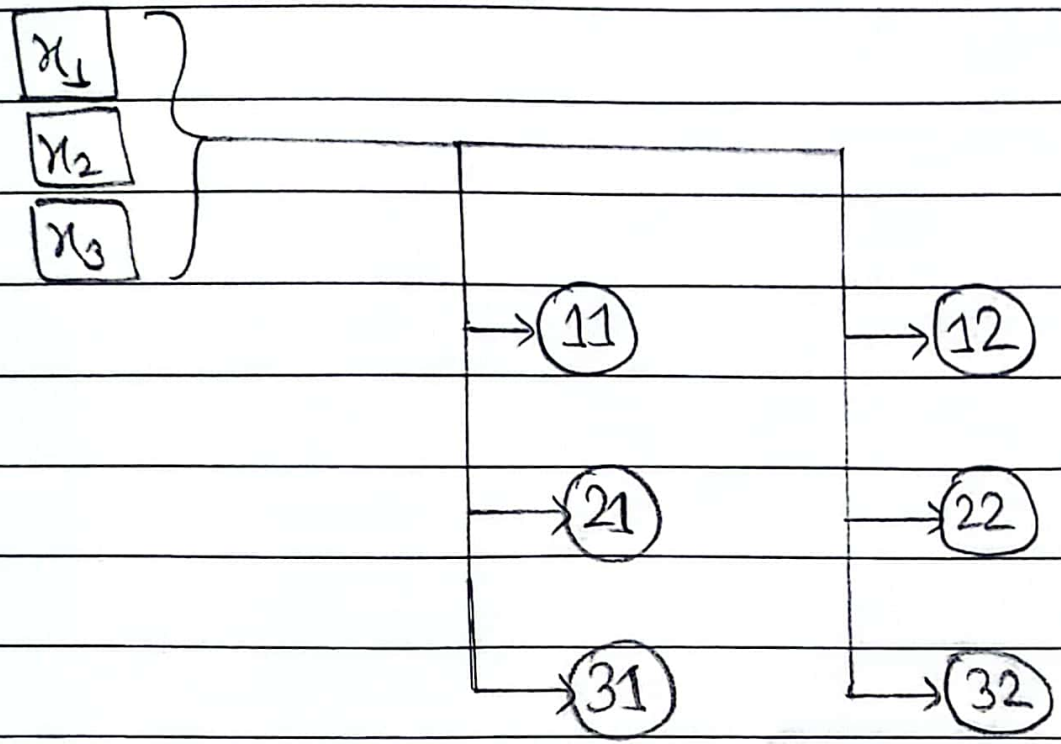
$$d_2 = d(x, w_2) = \sqrt{(0.5 - 0.14)^2 + (0.3 - 0.28)^2 + (0.7 - 0.198)^2} = 0.6$$

$$d_3 = d(x, w_3) = \sqrt{(0.5 - 0.3)^2 + (0.3 - 0.6)^2 + (0.7 - 0.5)^2} = 0.4$$

Clearly neuron 3 is winner.

Example:

Consider following 2-D SOM and 3-D inputs. Show the working of SOM for the inputs (0.25, 0.15, 0.55).



Initial weight matrix

0.1	0.2	0.4	0.2	0.1	0.2
0.4	0.3	0.2	0.5	0.7	0.6
0.7	0.5	0.6	0.3	0.4	0.3

classmate
Date
Page

Vector Quantization

- Vector quantization (VQ) is a lossy data compression technique.
- In VQ, a codebook is used to approximate each input vector.
- Every input vector is assigned to the nearest vector in the codebook.
- The quantization process can be decomposed into two operations : Encoder and Decoder.
- Encoder maps every input vector with some index and decoder maps every index with nearest code vector of the input vector.

Example:

Compress the input $\{(1,5),(2,3),(4,3),(4,1),(6,3),(5,4),(2,3),(2,4)\}$ using codebook $\{(1,1),(3,3),(5,5),(7,7)\}$

Input → Encoder (Data Compression) → Decoder (Decompression) → Output

Encoder (Data Compression)		
CodeBook		Index
1	1	00
3	3	01
5	5	10
7	7	11

Decoder (Decompression)		
Index	CodeBook	
00	1	1
01	3	3
10	5	5
11	7	7

Sol:

- Lets take input $x = (1,5)$

$d(x, c_1) = \sqrt{(1-1)^2 + (5-1)^2} = 4$

$d(x, c_2) = \sqrt{(1-3)^2 + (5-3)^2} = 2.83$

$d(x, c_3) = \sqrt{(1-5)^2 + (5-5)^2} = 4$

$d(x, C_4) = \sqrt{(1-7)^2 + (5-7)^2} = 6.32$

Thus, (1,5) is matched with (3,3) $\Rightarrow$ code for (1,5) is 01.

- Lets take input $x = (2,3)$

$d(x, C_1) = \sqrt{(2-1)^2 + (3-1)^2} = 2.24$

$d(x, C_2) = \sqrt{(2-3)^2 + (3-3)^2} = 1$

$d(x, C_3) = \sqrt{(2-5)^2 + (3-5)^2} = 3.61$

$d(x, C_4) = \sqrt{(2-7)^2 + (3-7)^2} = 6.40$

Thus, (2,3) is matched with (3,3) $\Rightarrow$ code for (2,3) is 01.

- Lets take input $x = (4,3)$

$d(x, C_1) = \sqrt{(4-1)^2 + (3-1)^2} = 3.61$

$d(x, C_2) = \sqrt{(4-3)^2 + (3-3)^2} = 1$

$d(x, C_3) = \sqrt{(4-5)^2 + (3-5)^2} = 2.24$

$d(x, C_4) = \sqrt{(4-7)^2 + (3-7)^2} = 5$

Thus, (4,3) is matched with (3,3) $\Rightarrow$ code for (4,3) is 01.

- Lets take input $x = (4,1)$

$d(x, C_1) = \sqrt{(4-1)^2 + (1-1)^2} = 3$

$d(x, C_2) = \sqrt{(4-3)^2 + (1-3)^2} = 2.24$

$d(x, C_3) = \sqrt{(4-5)^2 + (1-5)^2} = 4.12$

$d(x, C_4) = \sqrt{(4-7)^2 + (1-7)^2} = 6.71$

Thus, (4,1) is matched with (3,3) $\Rightarrow$ code for (4,1) is 01.

- Lets take input $x = (6,3)$

$d(x, C_1) = \sqrt{(6-1)^2 + (3-1)^2} = 5.39$

$d(x, C_2) = \sqrt{(6-3)^2 + (3-3)^2} = 3$

$d(x, C_3) = \sqrt{(6-5)^2 + (3-5)^2} = 2.24$

$d(x, C_4) = \sqrt{(6-7)^2 + (3-7)^2} = 4.12$

Thus, (6,3) is matched with (5,5) $\Rightarrow$ code for (6,3) is 10.

Date
Page

- Lets take input $x = (5,4)$

$d(x,c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$

$d(x,c_2) = \sqrt{(5-3)^2 + (4-3)^2} = 2.24$

$d(x,c_3) = \sqrt{(5-5)^2 + (4-5)^2} = 1$

$d(x,c_4) = \sqrt{(5-7)^2 + (4-7)^2} = 3.61$

Thus, (5,4) is matched with (5,5) $\Rightarrow$ code for (5,4) is 10.

- Lets take input $x = (2,3)$

$d(x,c_1) = \sqrt{(2-1)^2 + (3-1)^2} = 2.24$

$d(x,c_2) = \sqrt{(2-3)^2 + (3-3)^2} = 1$

$d(x,c_3) = \sqrt{(2-5)^2 + (3-5)^2} = 3.61$

$d(x,c_4) = \sqrt{(2-7)^2 + (3-7)^2} = 6.40$

Thus, (2,3) is matched with (3,3) $\Rightarrow$ code for (2,3) is 01.

- Lets take input $x = (2,4)$

$d(x,c_1) = \sqrt{(2-1)^2 + (4-1)^2} = 3.16$

$d(x,c_2) = \sqrt{(2-3)^2 + (4-3)^2} = 1.41$

$d(x,c_3) = \sqrt{(2-5)^2 + (4-5)^2} = 3.16$

$d(x,c_4) = \sqrt{(2-7)^2 + (4-7)^2} = 5.81$

Thus, (2,4) is matched with (3,3) $\Rightarrow$ code for (2,4) is 01.

Without VQ:

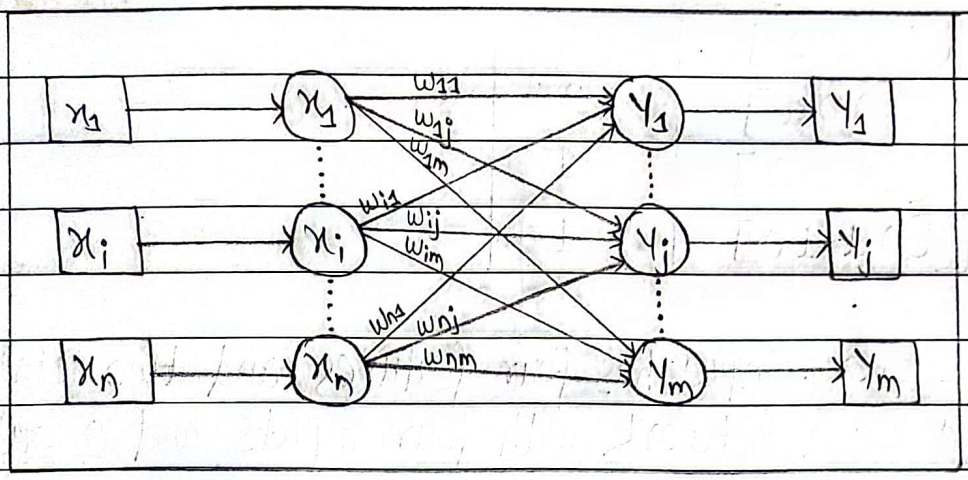
- We need 3 bits to represent 8 inputs.
- Thus, total number of bits $= 8 \times 3 = 24$ bits

After VQ:

- 2 bits are needed to represent 4 code vectors
- Thus, total number of bits $= 8 \times 2 = 16$ bits.

Learning Vector Quantization (LVQ).

- LVQ is prototype based supervised classification algorithm that uses competitive learning algorithm for training neural network.
- Training algorithm used by LVQ is similar to SOM except slight different in weight update.
- LVQ has two layers, one is the Input layer and the other one is the Output layer.
- The architecture of the Learning Vector Quantization with the m number of classes in an input data and n number of input features for any sample is given below:



Algorithm:

1. From the given set of training vectors, take the first m (i.e. number of clusters) training vectors and use them as weight vectors. The remaining vectors can be used for training.

2. For each training vector x
   - Calculate Euclidean distance between x and weight vector of each neuron.

$$D_j = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_i - w_{ij})^2$$

   - Find index of winning neuron (say, J) such that Euclidean distance is minimum.

3. Update weights of winning neuron as below.

If $T = J$ $\quad w_i = w_i + \alpha(x - w_i)$

If $T \neq J$ $\quad w_i = w_i - \alpha(x - w_i)$

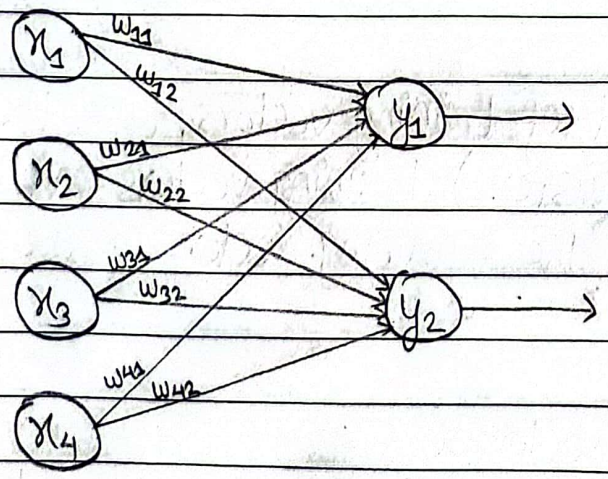
4. Repeat steps 2-3 until stopping condition is satisfied.

Example:

- Construct an LVQ net with five vectors assigned to two classes. Given vectors along with classes is shown below.

Vector	Class
[0 0 1 1]	1
[1 0 0 0]	2
[0 0 0 1]	2
[1 1 0 0]	1
[0 1 1 0]	1

Sol: Here every input has four features and two classes, so draw a network with four inputs and 2 outputs.



Initialize weight vector $w = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$ $\quad$ Let $\alpha = 0.1$

Take First Input Vector $x = \{0\ 0\ 0\ 1\}$ $T = 2$ (i.e. class 2)

Calculate Euclidean distances:

$D_1 = (0-0)^2 + (0-0)^2 + (0-1)^2 + (1-1)^2 = 1$

$D_2 = (0-1)^2 + (0-0)^2 + (0-0)^2 + (1-0)^2 = 2$

Thus, winning neuron is: $j = 1$ (i.e. y_1)

Update weights ($T \neq J$)

$w_{11} = w_{11} - \alpha(x_1 - w_{11}) = 0 - 0.1(0-0) = 0$

$w_{21} = w_{21} - \alpha(x_2 - w_{21}) = 0 - 0.1(0-0) = 0$

$w_{31} = w_{31} - \alpha(x_3 - w_{31}) = 1 - 0.1(0-1) = 1.1$

$w_{41} = w_{41} - \alpha(x_4 - w_{41}) = 1 - 0.1(1-1) = 1$

Thus, updated weight matrix is

$$w = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

Take Second Input Vector $x = \{1\ 1\ 0\ 0\}$ $T = 1$

Calculate Euclidean distances:

$D_1 = (1-0)^2 + (1-0)^2 + (0-1.1)^2 + (0-1)^2 = 4.21$

$D_2 = (1-1)^2 + (1-0)^2 + (0-0)^2 + (0-0)^2 = 1$

Thus, winning neuron is: $j = 2$ (i.e. y_2)

Update weights ($T \neq J$)

$w_{12} = w_{12} - \alpha(x_1 - w_{12}) = 1 - 0.1(1-1) = 1$

$w_{22} = w_{22} - \alpha(x_2 - w_{22}) = 0 - 0.1(1-0) = -0.1$

$w_{32} = w_{32} - \alpha(x_3 - w_{32}) = 0 - 0.1(0-0) = 0$

$w_{42} = w_{42} - \alpha(x_4 - w_{42}) = 0 - 0.1(0-0) = 0$

Thus, updated weight matrix is

$$w = \begin{bmatrix} 0 & 1 \\ 0 & -0.1 \\ 1.1 & 0 \\ 1 & 0 \end{bmatrix}$$

Take Third Input Vector $x = \{0\ 1\ 1\ 0\}$ $T = 1$

Calculate Euclidean distances:

$D_1 = (0-0)^2 + (1-0)^2 + (1-1.1)^2 + (0-1)^2 = 2.01$

$D_2 = (0-1)^2 + (1+0.1)^2 + (1-0)^2 + (0-0)^2 = 3.21$

Thus, winning neuron is: $j=1$ (i.e. y_1).

Update weights ($T = 1$)

$w_{11} = w_{11} + \alpha(x_1 - w_{11}) = 0 + 0.1(0-0) = 0$

$w_{12} = w_{12} + \alpha(x_2 - w_{12}) = 0 + 0.1(1-0) = 0.1$

$w_{13} = w_{13} + \alpha(x_3 - w_{13}) = 1.1 + 0.1(1-1.1) = 1.09$

$w_{14} = w_{14} + \alpha(x_4 - w_{14}) = 1 + 0.1(0-1) = 0.9$.

Thus, weight matrix is

$$\begin{bmatrix} 0 & 1 \\ 0.1 & -0.1 \\ 1.09 & 0 \\ 0.9 & 0 \end{bmatrix}$$

Unit-7

Recurrent Neural Network

Introduction to RNN

- Recurrent Neural Network(RNN) are a type of neural Network where the output from previous step are fed as input to the current step.
- In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words.

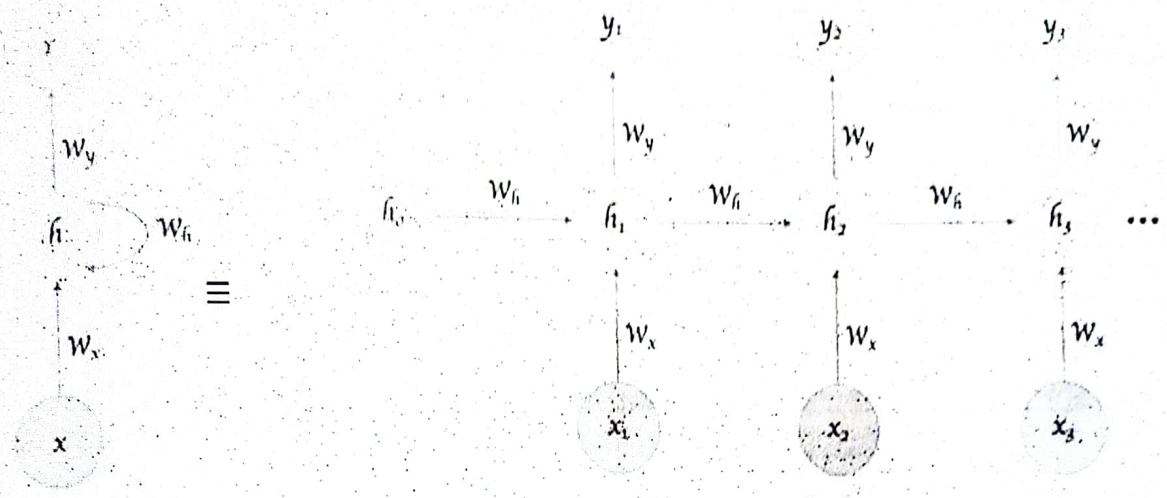
Introduction to RNN

- Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.
- RNN have a **"memory"** which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden states to produce the output.

Introduction to RNN

- Thus, RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous outputs by giving each output as input to the next hidden layer (see right part of the figure in next slide).
- Hence layers of neural network in right side can be joined together such that the weights and bias of all the hidden layers is the same, into a single recurrent layer.

Introduction to RNN



Computation of hidden state

$$h_t = f(w_h h_{t-1} + w_x x_t)$$

Computation of output

$$y_t = f(w_y h_t)$$

Introduction to RNN

Applications of RNN

- Time series prediction
- Sequence prediction
- Natural Language Processing
- Speech processing

Forward Propagation in RNN

Example: Consider the word 'dogs'. Show forward propagation of RNN with three nodes in hidden layer to predict letter **'s'** given the letters **'d'**, **'o'**, and **'g'**. Assume that hidden layer activation function is Tanh and activation in output layer is softmax.

Solution

One hot encoding of the input: input vocabulary{d, o, g, s}

d	o	g	s
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Forward Propagation in RNN

We know that:

$$h_t = f(w_h h_{t-1} + w_x x_t) = \tanh(w_h h_{t-1} + w_x x_t) \quad Note: Soft\max(x) = \frac{e^x}{\sum_{i=1}^{n} e^{x_i}}$$

$$y_t = f(w_y h_t) = soft\max(w_y h_t)$$

Here, dimension of w_h is d × d, dimension of w_x is d × k, and dimension of w_y is k × d. Where, d is number of hidden nodes, and k is dimension of input vector.

Assume that:

$$w_h = \begin{bmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{bmatrix} \quad w_x = \begin{bmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.9 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{bmatrix} \quad w_y = \begin{bmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{bmatrix}$$

Forward Propagation in RNN

Thus, (for input 'd')

$$h_t = \tanh\left(\begin{bmatrix} 0.1 & 0.5 & 0.1 \\ 0.5 & 0.9 & 0.3 \\ 0.3 & 0.2 & 0.1 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.6 & 0.8 & 0.4 & 0.8 \\ 0.2 & 0.9 & 0.8 & 0.7 \\ 0.9 & 0.8 & 0.1 & 0.2 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}\right) = \tanh\left(\begin{bmatrix} 0.6 \\ 0.2 \\ 0.9 \end{bmatrix}\right) = \begin{bmatrix} 0.54 \\ 0.20 \\ 0.72 \end{bmatrix}$$

$$y_t = soft\max\left(\begin{bmatrix} 0.9 & 0.8 & 0.3 \\ 0.2 & 0.3 & 0.4 \\ 0.6 & 0.9 & 0.1 \\ 0.5 & 0.0 & 0.3 \end{bmatrix}\begin{bmatrix} 0.54 \\ 0.20 \\ 0.72 \end{bmatrix}\right) = soft\max\left(\begin{bmatrix} 0.86 \\ 0.45 \\ 0.57 \\ 0.48 \end{bmatrix}\right) = \begin{bmatrix} 0.32 \\ 0.21 \\ 0.24 \\ 0.22 \end{bmatrix}$$

At t=1, RNN predicts the letter "d" given the input "d". This doesn't make sense but it's ok because we've used untrained random weights.

Forward Propagation in RNN

Example: Consider the input series {0.2,0.4,0.6, 0.8}. Show forward propagation of RNN with two nodes in hidden layer to predict next term in the series. Assume that activation function of hidden layer is sigmoid and activation in output layer is linear.

Solution {Input: 0.2 }

$$h_t = \tanh\left(\begin{bmatrix} 0.1 & 0.5 \\ 0.5 & 0.9 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}[0.2]\right) = \sigma\left(\begin{bmatrix} 0.12 \\ 0.04 \end{bmatrix}\right) = \begin{bmatrix} 0.53 \\ 0.51 \end{bmatrix}$$

$$y_t = \begin{bmatrix} 0.4 & 0.7 \end{bmatrix}\begin{bmatrix} 0.53 \\ 0.51 \end{bmatrix} = 0.57$$

Computational Power of recurrent Networks

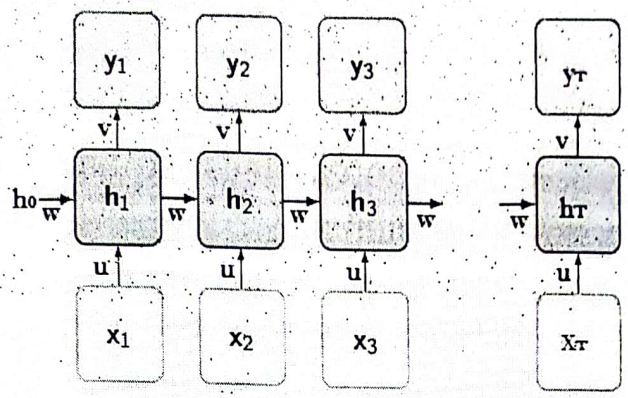
- Computational power of recurrent neural network is based on following theorem.
- Theorem: *All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions* (**(Siegelmann and Sontag, 1991)**.
- They show that, for every computable function, there exists a finite RNN (of the form described above) that can compute it. They do this by showing that it's possible to use a RNN to explicitly simulate a pushdown automaton with two stacks.

Computational Power of recurrent Networks

- This is another model that's computationally equivalent to a Turing machine. Any computable function can be computed by a Turing machine. Any Turing machine can be simulated by a pushdown automaton with two stacks.
- Any pushdown automaton with two stacks can be simulated by a RNN. Therefore, any computable function can be computed by a RNN.

Backpropagation Through Time (BPTT)

- BPTT is gradient based algorithm for training recurrent neural networks.
- Unrolling of RNN with T layers looks like below, where w, u, v are weigh matrices.



Backpropagation Through Time (BPTT)

- Let L is total loss in the RNN. Then the weights w, u, v are updated as below.

$$w = w - \alpha \frac{\partial L}{\partial w}$$

$$v = v - \alpha \frac{\partial L}{\partial v}$$

$$u = u - \alpha \frac{\partial L}{\partial u}$$

- Thus, we need to compute $\partial L/\partial w, \partial L/\partial v, \partial L/\partial u$ for updating weights.

Backpropagation Through Time (BPTT)

- Let L_t is the loss at t^{th} step of RNN. Then total loss of the RNN is given by.

$$L = \sum_{t=1}^{T} L_t$$

- Loss function can be different according to need. Basically, cross-entropy or least square loss function is used. Normally, Cross-entropy loss function is used for classification problems whereas least square loss function is used for regression problems.

Backpropagation Through Time (BPTT)

- Recall that computation of hidden state and output in RNN is performed as below:

Computation of hidden state *Computation of output*

$$h_t = f(wh_{t-1} + ux_t) \qquad y_t = g(vh_t)$$

- Recall that, we need to compute $\partial L/\partial w, \partial L/\partial v, \partial L/\partial u$ for BPTT. Assume that t=3

$$h_3 = f(wh_2 + ux_3) \qquad y_3 = g(vh_3)$$

- Also, assume that

$$L_3 = \frac{1}{2}(d_3 - y_3)^2$$

, where d is desired output and y is predicted output

Backpropagation Through Time (BPTT)

- Now,

$$\frac{\partial L_3}{\partial v} = \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial v} = -(d_3 - y_3)h_3$$

$$\frac{\partial L_3}{\partial w} = \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial w} = -(d_3 - y_3)vf'(z_3)\frac{\partial z_3}{\partial w}$$

$$= -(d_3 - y_3)vf'(z_3)\left[h_2 + w\frac{\partial h_2}{\partial w}\right]$$

- Here, $z_3 = wh_2 + ux_3$
- In the above relation derivative of h_2 needs to be computed recursively in terms of h_1.

Backpropagation Through Time (BPTT)

- Finally,

$$\frac{\partial L_3}{\partial u} = \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial u} = -(d_3 - y_3)vf'(z_3)\frac{\partial z_3}{\partial u}$$

$$= -(d_3 - y_3)vf'(z_3)\left[x_3 + w\frac{\partial h_2}{\partial u}\right]$$

- Again, in the above relation, derivative of h_2 needs to be computed recursively in terms of h_1.
- Since the derivatives $\partial L/\partial v$, and $\partial L/\partial u$ needs to be calculated recursively by back propagating through time the algorithm is named as BPTT.

Backpropagation Through Time (BPTT)

Algorithm

1. For i=1 to T Compute y_i and L_i as below
   $y_i = g(vh_i)$, *where v is weight matrix*
   $L_i = \frac{1}{2}(d_i - y_i)^2$, *where d is desired output and y is predicted output*
2. For i=1 to T Compute following gradients
   $\partial L_i/\partial w, \partial L_i/\partial v, \partial L_i/\partial u$
3. Find total gradient by summing all local gradients
4. Update weights as below

$$w = w - \alpha\frac{\partial L}{\partial w} \qquad v = v - \alpha\frac{\partial L}{\partial v} \qquad u = u - \alpha\frac{\partial L}{\partial u}$$

Vanishing Gradient and Truncated BPTT

- Two main problems associated with Recurrent Neural networks are:
  1. Gradient calculations either vanish or explode
  2. Gradient calculations are expensive
- Solution to exploding gradient is gradient clipping and solution to vanishing gradient problem is to use alternate architectures like gated recurrent unit (GRU) network and Long Short-term Memory (LSTM) network. Both, GRU and LSTM are variants of RNN architecture.
- Solution to second problem is to used truncated BPTT algorithm

Vanishing Gradient and Truncated BPTT

- RNNs might have a difficulty in learning long range dependencies. For instance if we have a sentence like "The man who ate my pizza has purple hair". In this case, the description purple hair is for the man and not for the pizza. So this is a long dependency.
- In BPTT algorithm, we need to compute gradient of loss to update weights.
- In order to compute loss we need to use recursive computation.
- For instance following two gradients can be computed as below.

Vanishing Gradient and Truncated BPTT

$$\frac{\partial L_t}{\partial w} = -(d_t - y_t) v f'(z_t) \left[h_{t-1} + w \frac{\partial h_{t-1}}{\partial w} \right]$$

$$\frac{\partial L_t}{\partial u} = -(d_t - y_t) v f'(z_t) \left[x_t + w \frac{\partial h_{t-1}}{\partial u} \right]$$

- In the above equations gradient of h_{t-1} needs to be computed recursively in terms of h_{t-2} . Thus, the weight matrix w is multiplied by many times while computing the gradients.

Vanishing Gradient and Truncated BPTT

- If w is small, we experience a vanishing gradient problem. And if w is large, we experience an exploding gradient problem.
- The lower the gradient is, the harder it is for the network to update the weights and the longer it takes to get to the final result.
- Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. This has the effect of our model being unstable and unable to learn from your training data.

Vanishing Gradient and Truncated BPTT

- In BPTT algorithm we need to back propagate through entire sequence in order to compute gradient of local loss. Thus, computation of gradient of loss is time consuming process.
- The practical solution is limit the backpropagation to a maximum of window of M steps. The forward pass should still be performed over the entire sequence, but the backward pass is truncated to windows of size M.
- This modified version of BPTT algorithm is called Truncated BPTT or TBPTT.

Vanishing Gradient and Truncated BPTT

The TBPTT algorithm can be summarized as below:

1. Present a sequence of k1 time steps of input and output pairs to the network.
2. calculate and accumulate errors across k2 time steps.
3. update weights.
4. Repeat

As we can clearly see that you need two parameters namely k1 and k2 for implementing TBPTT. K1 is the number of forward pass time steps between updates. This influences how fast or slow will be the training and the frequency of the weight updates.

Vanishing Gradient and Truncated BPTT

- On the other hand, k2 is the number of time steps which apply to BPTT. It should be large enough to capture the temporal structure in the problem for the network to learn.
- But the problem with TBPTT is that the network can't learn long dependencies as in BPTT because of limit on flow of gradient due to truncation.

Long Short-term Memory

- Because of the vanishing gradient problem, RNNs cannot remember long-term dependencies in time series data.
- LSTM is an RNN variant designed to address this issue. LSTM, like RNN, has a hidden state that serves as short-term memory.
- Furthermore, it maintains cell state, which serves as long-term memory.

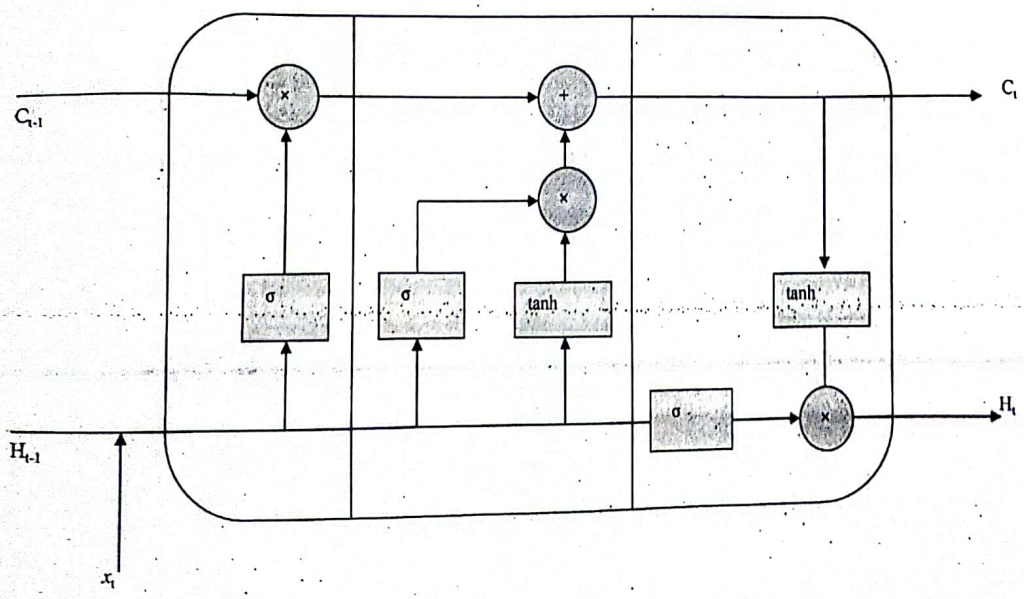
Long Short-term Memory

- The LSTM architecture is made up of three gates: *a forget gate, an input gate, and an output gate.*
- The forget gate is in charge of determining what information must be remembered and what can be forgotten from the previous state.
- The input gate determines which information from the current input is stored in the cell state.

Long Short-term Memory

- Finally, the output gate determines the next hidden state value.
- The LSTM network's architecture is depicted in the Figure and mathematical formulation is given in Equation given in next slide..

Long Short-term Memory



Long Short-term Memory

Forget Gate

$$F_t = \sigma(x_t W_f + H_{t-1} U_f)$$

Input Gate

$$I_t = \sigma(x_t W_i + H_{t-1} U_i) \quad (Input\ gate)$$

$$H'_t = tanh(x_t W_c + H_{t-1} U_c) \quad (New\ Information)$$

$$C_t = (C_{t-1} \times F_t + I_t \times H'_t) \quad (New\ Cell\ State)$$

Output Gate

$$O_t = \sigma(x_t W_o + H_{t-1} U_o)$$

$$H_t = tanh(C_t) \times O_t)$$

Long Short-term Memory

- Here, W_f, W_i, W_o, U_f, U_i, and U_o are weight matrices. x_t is input at time step t, H_t is hidden state at time t, C_t is cell state at time t, O_t is output at time t, and H'_t is candidate hidden state at time t. × is point-wise multiplication and + is point-wise addition.

Gated Recurrent Unit Network

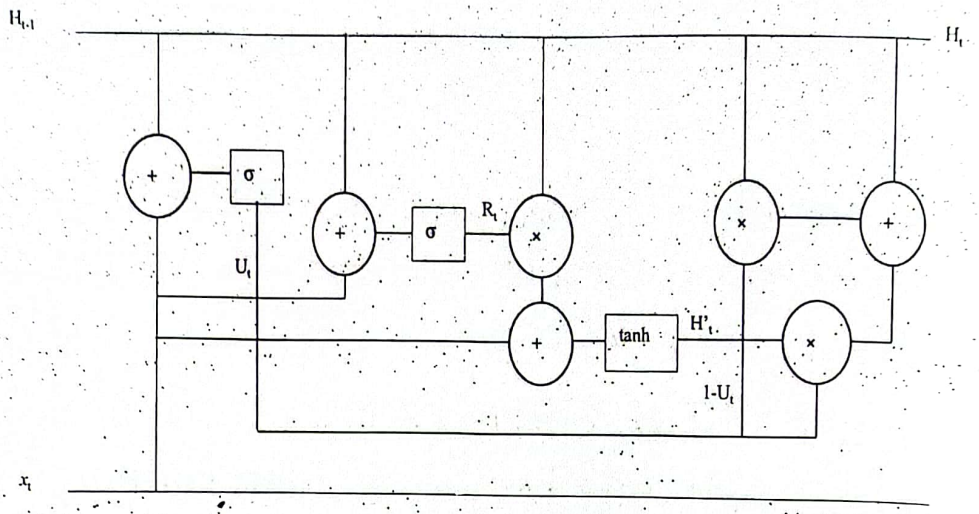
- GRU is another popular RNN variant that is used to solve the vanishing gradient problem.
- GRU, like LSTM, employs a gating mechanism to remember long-term dependencies. However, its structure is simpler than that of LSTM.
- As a result, training takes less time. GRU networks have been shown in studies to perform similarly to LSTM networks.
- Unlike LSTM, GRU does not maintain a separate cell state. It only keeps a hidden state.

Gated Recurrent Unit Network

- The network only has two gates: an update gate and a reset gate.
- The reset gate controls how much of the previous knowledge is erased. It is in charge of the network's short-term memory.
- The update gate determines how much past knowledge must be passed on into the future. It is in charge of the network's long-term memory.
- Finally, the results of both gates are used to compute the network's current hidden state.

Gated Recurrent Unit Network

- The network's architecture is depicted in the Figure given below, and its mathematical formulation is given in the Equation given in next slide.



Gated Recurrent Unit Network

$$R_t = \sigma(x_t W_r + H_{t-1} V_r)$$
$$U_t = \sigma(x_t W_u + H_{t-1} V_u)$$
$$H_t' = tanh(x_t W_h + V_h(R_t \times H_{t-1}))$$
$$H_t = (U_t \times H_t') + (1 - U_t) \times H_{t-1})$$

Where, R_t is reset gate and U_t is update gate. H'_t and H_t are candidate hidden state and hidden state. W and V are weight matrices. × is element-wise multiplication.

Hopfield Networks

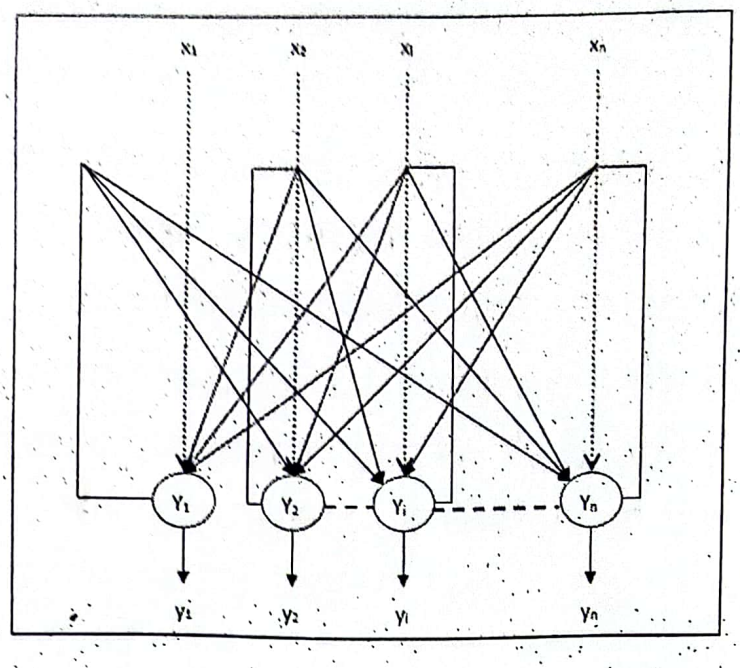
- Hopfield networks are single layer recurrent neural networks that consists of n fully connected neurons.
- This networks acts as associative memory of content addressable memory and is used for different pattern recognition problem.
- An associative memory stores a set of p patterns in such a way that when presented with a new pattern, the network responds by producing whichever one of the stored patterns most closely resembles.

Hopfield Networks

- In this network every neuron acts input as well as output node.
- No connection from a node to itself is allowed in this type of neural network.
- Hopfield nets can be discrete or continuous. Furthermore, Discrete Hopfield nets can be binary or bipolar.
- Binary Hopfield nets produce 0 or 1 output whereas bipolar nets produce 1 or -1 output.

Hopfield Networks

- Architecture of Hopfield network is given below.



Hopfield Networks

- In this network every neuron acts input as well as output node.
- No connection from a node to itself is allowed in this type of neural network.
- Hopfield nets can be discrete or continuous. Furthermore, Discrete Hopfield nets can be binary or bipolar.
- Binary Hopfield nets produce 0 or 1 output whereas bipolar nets produce 1 or -1 output.

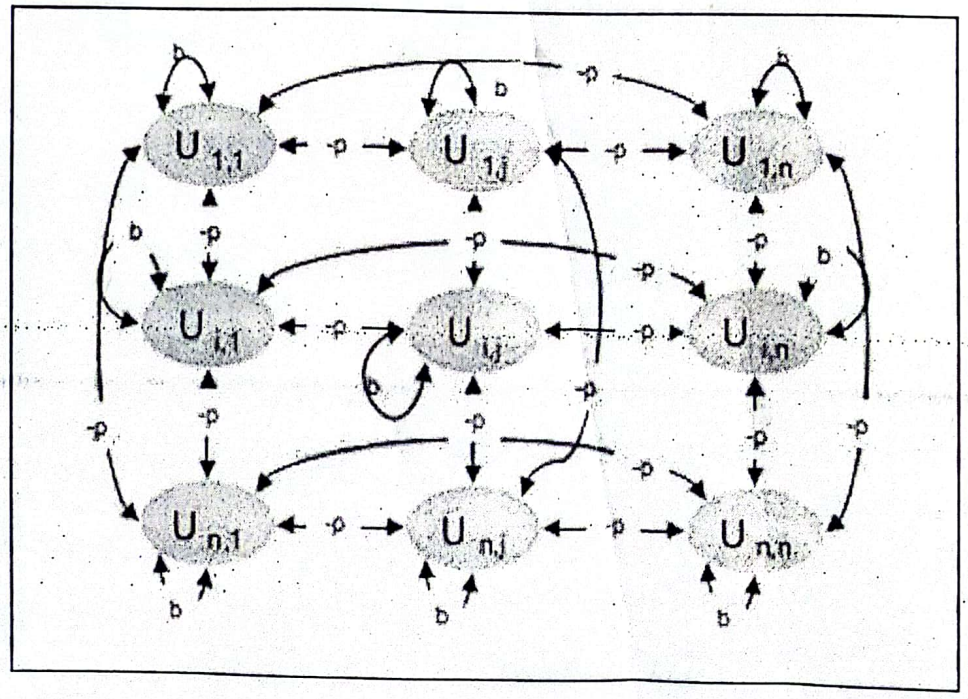
Boltzmann Machine

- Boltzmann Machine is a kind of recurrent neural network where the nodes make binary decisions and are present with certain biases.
- It consists of a network of symmetrically connected, neuron-like units that make decisions stochastically whether to be active or not.
- Boltzmann machines have a simple learning algorithm that allows them to discover interesting features in datasets composed of binary vectors.

Boltzmann Machine

- Some important points about Boltzmann Machine.
  - They use recurrent structure.
  - They consist of stochastic neurons, which have one of the two possible states, either 1 or 0.
  - Some of the neurons in this are adaptive and some are clamped.
  - If we apply simulated annealing on discrete Hopfield network, then it would become Boltzmann Machine.

Boltzmann Machine

- The main purpose of Boltzmann Machine is to optimize the solution of a problem. It is the work of Boltzmann Machine to optimize the weights and quantity related to that particular problem.
- The following diagram shows the architecture of Boltzmann machine. It is clear from the diagram, that it is a two-dimensional array of units. Here, weights on interconnections between units are **–p** where **p > 0**. The weights of self-connections are given by **b** where **b > 0**.

Boltzmann Machine