IFET Autonomous College of Engineering
Permenantly Affiliated to Anna University|Approved by AICTE

DEPARTMENT OF INFORMATION TECHNOLOGY
YEAR/SEM: III/VI
23CSX502 – DEEP LEARNING AND NEURAL NETWORKS
SOLVED

UNIT I
INTRODUCTION TO NEURAL NETWORK

Neural Networks Basics - Functions in Neural networks – Classification and Clustering problems - Deep networks basics - Shallow neural networks – Deep Neural Networks – Forward and Back Propagation. Deep Learning Frameworks – Data Augmentation - Underfitting- Overfitting.

PART- B

2 **Compare supervised, unsupervised, and reinforcement learning with examples.** **16** **K3**

Supervised, Unsupervised, and Reinforcement Learning

Machine Learning is a subset of Artificial Intelligence (AI) that enables computers to learn from data and make decisions without explicit programming. It is broadly categorized into **Supervised Learning, Unsupervised Learning, and Reinforcement Learning**.

1. Supervised Learning

Supervised learning is a type of machine learning where the model is trained using **labeled data**. Each input data has a corresponding correct output, allowing the model to learn a mapping function between input and output.

The goal is to make accurate predictions when given new, unseen data.

Examples:

1. **Classification** – Predicting if an email is spam or not.
2. **Regression** – Predicting house prices based on features like location and size.

Common Algorithms:

- **Linear Regression** – Used for predicting continuous values.
- **Decision Trees** – Used for both classification and regression.
- **Support Vector Machines (SVM)** – Used for separating data into different categories.
- **Neural Networks** – Used for complex pattern recognition.

Real-World Applications:

- Fraud detection in banking.
- Medical diagnosis (predicting diseases).
- Image recognition (identifying objects in photos).

Analogy: A student learning from a teacher with correct answers provided for reference.

2. Unsupervised Learning

Unsupervised learning is a type of machine learning where the model is trained on **unlabeled data**. There are no predefined outputs, and the algorithm identifies hidden patterns and relationships within the data.

The goal is to find structure **in the data, such as grouping similar data points or reducing complexity.**

Examples:

1. **Clustering** – Grouping customers based on purchasing behavior.
2. **Anomaly Detection** – Detecting fraudulent transactions in banking.
3. **Dimensionality Reduction** – Reducing data complexity for visualization.

Common Algorithms:

- **K-Means Clustering** – Groups data into clusters based on similarity.
- **Hierarchical Clustering** – Builds a tree-like structure of clustered data.
- **Principal Component Analysis (PCA)** – Reduces the number of variables in a dataset while preserving key information.

Real-World Applications:

- Market segmentation in business.
- Organizing news articles based on topic.
- Identifying criminal activities from transaction patterns.

Analogy: A student sorting books in a library without knowing the categories beforehand.

3. Reinforcement Learning (RL)

Reinforcement learning is a machine learning approach where an **agent learns by interacting with an environment** and receiving rewards or penalties based on its actions.

The goal is to maximize cumulative rewards by learning the best sequence of actions over time.

Key Components:

- **Agent** – The learner or decision-maker (e.g., a robot, AI in a game).
- **Environment** – The world in which the agent operates.
- **Actions** – The moves the agent can make.
- **Reward** – The feedback from the environment (positive or negative).

Examples:

1. **Game AI** – Training an AI to play chess or Go.
2. **Self-Driving Cars** – Learning how to navigate safely.
3. **Robotics** – Teaching robots to walk or complete tasks.

Common Algorithms:

- **Q-Learning** – A model-free algorithm for learning optimal policies.
- **Deep Q Networks (DQN)** – Uses deep learning to improve Q-Learning.
- **Policy Gradient Methods** – Directly optimize policies for better actions.

Real-World Applications:

- Autonomous robots in industries.
- Stock market trading algorithms.
- AI assistants improving responses over time.

Analogy: Training a pet using rewards (treats) and punishments to shape behavior.

Table 1.4 Difference Between Supervised, Unsupervised, and Reinforcement Learning

Feature	Supervised Learning	Unsupervised Learning	Reinforcement Learning
1. Definition	Learns from **labeled data** (input-output pairs).	Learns from **unlabeled data**, finding hidden patterns.	Learns through **trial and error**, receiving rewards or penalties.
2. Objective	Predict outcomes for new data.	Identify structures, clusters, or relationships in data.	Take optimal actions to maximize long-term rewards.
3. Training Data	Labeled (contains correct answers).	Unlabeled (no predefined correct answers).	No predefined dataset; learns from experience.
4. Learning Type	**Direct** – Mapping input to output.	**Indirect** – Identifies patterns and groups.	**Interactive** – Learns by interacting with an environment.

5. Example Tasks	Classification (spam detection), Regression (house price prediction).	Clustering (customer segmentation), Anomaly detection (fraud detection).	Game playing (chess AI), Robotics (self-driving cars).
6. Common Algorithms	Linear Regression, SVM, Neural Networks.	K-Means, Hierarchical Clustering, PCA.	Q-Learning, Deep Q Networks (DQN), Policy Gradients.
7. Output Type	A specific prediction (label or value).	A structure or grouping in data.	A sequence of actions (policy) for decision-making.
8. Dependency on Feedback	Uses direct feedback (correct labels).	No feedback; finds its own structure.	Receives delayed rewards or penalties.
9. Application Fields	Healthcare, finance, email filtering.	Market research, anomaly detection, recommendation systems.	Robotics, gaming, stock trading.
10. Human Involvement	Requires labeled datasets from humans.	Less human involvement; data patterns emerge naturally.	Requires designing a reward system but learns autonomously.
11. Computation Complexity	Moderate to high (depends on dataset size).	Lower than supervised (no need for labeling).	High (requires continuous learning and interaction).
12. Analogy	A student learning from a teacher with correct answers provided.	A student sorting books without predefined categories.	A child learning to ride a bicycle through trial and error.

4 **a. Analyze the process of training hidden layers by ReLU in Deep Networks.** **8** **K3**

ReLU

- Rectified Linear Unit (ReLU) solve the vanishing gradient problem. ReLU is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero.
- It is the most commonly used activation function in neural networks, especially Convolutional Neural Networks (CNNs) and Multilayer perceptron's.
- . Mathematically, it is expressed as

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

  where X : input to neuron

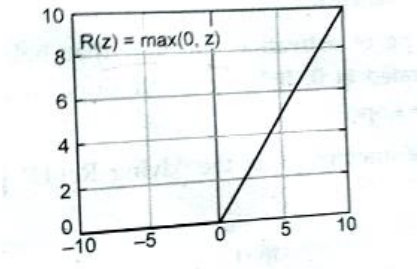


Figure 1.21: ReLU function

- The derivative of an activation function is required when updating the weights during the back-propagation of the error. The slope of ReLU is 1 for positive values and 0 for negative values.

It becomes non-differentiable when the input x is zero, but it can be safely assumed to be zero and causes no problem in practice.

- ReLU is used in the hidden layers instead of Sigmoid or tanh. The ReLU function solves the problem of computational complexity of the Logistic Sigmoid and Tanh functions.
- A ReLU activation unit is known to be less likely to create a vanishing gradient problem because its derivative is always 1 for positive values of the argument.

Advantages of ReLU function:

- ReLU is simple to compute and has a predictable gradient for the backpropagation of the error.
- Easy to implement and very fast.
- The calculation speed is very fast. The ReLU function has only a direct relationship.
- It can be used for deep network training.

Disadvantages of ReLU function:

- When the input is negative, ReLU is not fully functional which means when it comes to the wrong number installed, ReLU will die. This problem is also known as the Dead Neurons problem.
- b) ReLU function can only be used within hidden layers of a Neural Network Model.

b. Illustrate the working of backpropagation algorithm with an example. **8** **K3**

Backpropagation

Backpropagation is a training method used for a multi-layer neural network. It is also called the generalized delta rule. It is a gradient descent method which minimizes the total squared error of the output computed by the net.

The backpropagation algorithm looks for the minimum value of the error function in weight space using a technique called the delta rule or gradient descent. The weights that minimize the error function is then considered to be a solution to the learning problem.

Backpropagation is a systematic method for training multiple layer ANN. t is a generalization of Window-Hoff error correction rule. 80 % of ANN applications uses back propagation.

Consider a simple neuron:

a. Neuron has a summing junction and activation function.

b. Any non linear function which differentiable everywhere and increases everywhere with sum can be used as activation function.

C. Examples: Logistic function, Arc tangent function, hyperbolic tangent activation function,

These activation functions make the multilayer network to have greater representational power than single layer network only when non-linearity is introduced.

Need of hidden layers

- A network with only two layers (input and output) can only represent the input with whatever representation already exists in the input data.
- If the data is discontinuous or non-linearly separable, the innate representation is inconsistent, and the mapping cannot be learned using two layers (Input and Output).
- Therefore, hidden layer(s) are used between input and output layers Weights connects unit (neuron) in one layer only to those in the next higher layer.
- The output of the unit is scaled by the value of the connecting weight, and it is fed forward to

provide a portion of the activation for the units in the next higher layer.

- Backpropagation can be applied to an artificial neural network with any number of hidden layers. The training objective is to adjust the weights so that the application of a set of inputs produces the desired outputs.

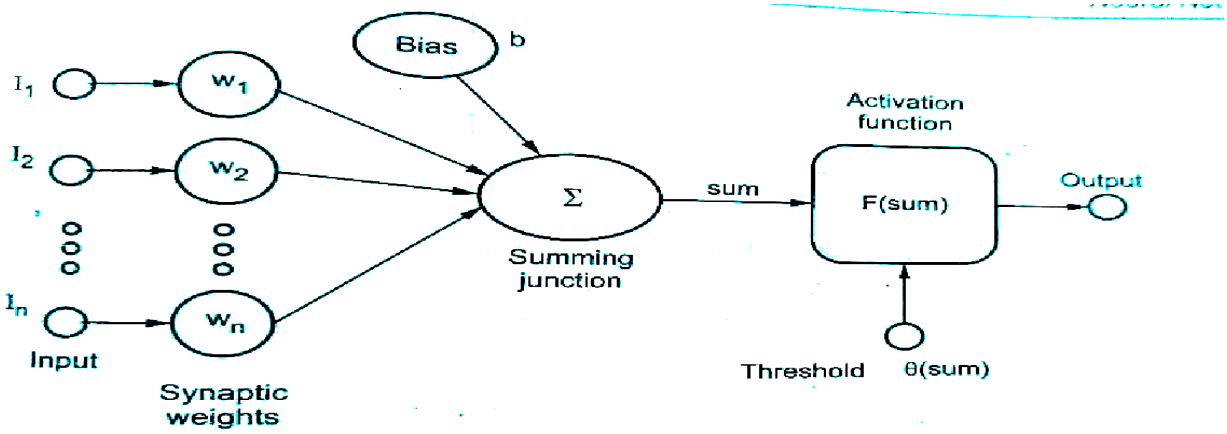


Figure 1.26: Back Propagation network

Training procedure: The network is usually trained with a large number of inputs - output pairs.

1. Generate weights randomly to small random values (both positive and negative) to ensure that the network is not saturated by large values of weights.
2. Choose a training pair from the training set.
3. Apply the input vector to network input.
4. Calculate the network output.
5. Calculate the error, the difference between the network output and the desired output.
6. Adjust the weights of the network in a way that minimizes this error.
7. Repeat steps 2 6 for each pair of input output in the training set until the error for the entire system is acceptably low.

Forward pass and backward pass:

• Backpropagation neural network training involves two passes.

1. In the forward pass, the input signals moves forward from the network input to the output.
2. In the backward pass, the calculated error signals propagate backward through the network, where they are used to adjust the weights.
3. In the forward pass, the calculation of the output is carried out, layer by layer in the forward direction. The output of one layer is the input to the next layer.

• In the reverse pass,

- The weights of the output neuron layer are adjusted first since the target value of each output neuron is available to guide the adjustment of the associated weights, using the delta rule.
- Next, we adjust the weights of the middle layers. As the middle layer neurons have no target values, it makes the problem complex.

Selection of number of hidden units: The number of hidden units depends on the number of input units.

1. Never choose h to be more than twice the number of input units.
2. You can load patterns of I elements into log, p hidden units.
3. Ensure that we must have at least 1/e times as many training examples.

4. Feature extraction requires fewer hidden units than inputs.
5. Learning many examples of disjointed inputs requires more hidden units than inputs.
6. The number of hidden units required for a classification task increases with the number of classes in the task. Large networks require longer training times.

Factor influencing Backpropagation training:

The training time can be reduced by using:

Bias : Networks with biases can represent relationships between outputs more easily than networks it out biases, Adding a bias to eat neuron 1s usually desirable to offset the origin of the activation function. The weight of the bias is trainable similar to weight except that the input is always +1.

1. **Momentum**: The use of momentum enhances the stability of the training process. Momentum is used to keep the training process going in the same general direction analogous to the way that momentum of a moving object behaves. In back propagation with momentum, the weight change is a combination of the current gradient and the previous gradient.

Advantages of backpropagation:

1. It is simple, fast and easy to program.
2. Only numbers of the input are tuned and not any other
3. No need to have prior knowledge about the network.
4. It is flexible.
5. A standard approach and works efficiently.
6. It does not require the user to learn special functions.

Disadvantages of backpropagation:

- ✓ Backpropagation possibly be sensitive to noisy data and irregularity.
- ✓ The performance of this is highly reliant on the input data.
- ✓ Needs excessive time for training.
- ✓ The need for a matrix-based method for backpropagation instead of mini - batch.

6	**Evaluate the use of regularization in machine learning and justify the difference between L1 and L2 regularization.**	**16**	**K4**

Regularization Techniques

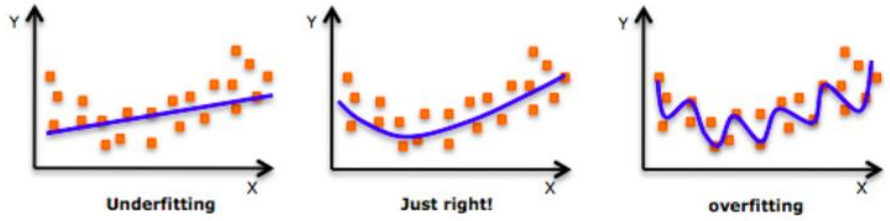


Figure 1.27: Regularization Techniques

- Just have a look at the above figure, and we can immediately predict that once we try to cover every minutest feature of the input data, there can be irregularities in the extracted features, which can introduce noise in the output. This is referred to as "Overfitting".
- This may also happen with the lesser number of features extract as some of the important details might be missed out. This will leave an effect on the accuracy of the outputs produced. This is referred to as "Underfitting".
- This also shows that the complexity for processing the input elements increases with overfitting. Also, neural networks being a complex interconnection of nodes the issue of overfitting may arise frequently.
- To eliminate this, regularization is used, in which we have to make the slightest modification in

the design of the neural network, and we can get better outcomes.

Regularization in Machine Learning

- ✓ One of the most important factors. that affect the machine learning model is overfitting.
- ✓ The machine learning model may perform poorly if it tries to capture even the noise present in the dataset applied for training the system, which ultimately results in overfitting. In this context, noise doesn't mean the ambiguous or false data, but those inputs which do not acquire the required features to execute the machine learning model.

Let this be the simple relation for linear regression:

$$Y = bo + bX + b,X,t\,.bp$$

Y = Learned relation

B(beta) = Co-efficient estimators for different variables and/or predictors (X)

- ✓ Now, we shall introduce a loss function that implements the fitting procedure, which is referred to as "Residual Sum of Squares'" or RSS.
- ✓ The co-efficient in the function is chosen in such a way that it can minimize the loss function easily.

$$\text{RSS}=\sum_{i=1}^{n}(Yi-\beta 0-\sum_{j=1}^{p}\beta jXij)2$$

• In case noise is present in the training dataset, then the adjusted co-efficient won't be generalized when the future datasets will be introduced. Hence, at this point, regularization comes into picture and makes this adjusted co-efficient shrink towards zero.

One of the methods to implement this is the ridge regression, also known as L2 regularization. Let's have a quick overview on this.

How does Regularization help reduce Overfitting?

Let's consider a neural network which is overfitting on the training data as shown in the image below.

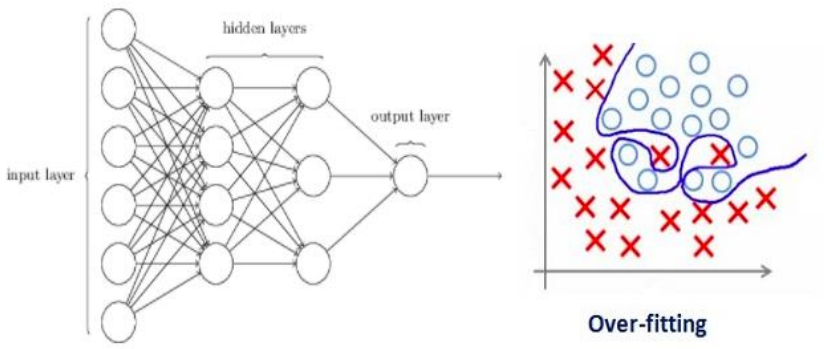


Figure 1.28: Regularization helps to reduce Overfitting

If you have studied the concept of regularization in machine learning, you will have a fair idea that regularization penalizes the coefficients. In deep learning, it actually penalizes the weight matrices of the nodes.

Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.

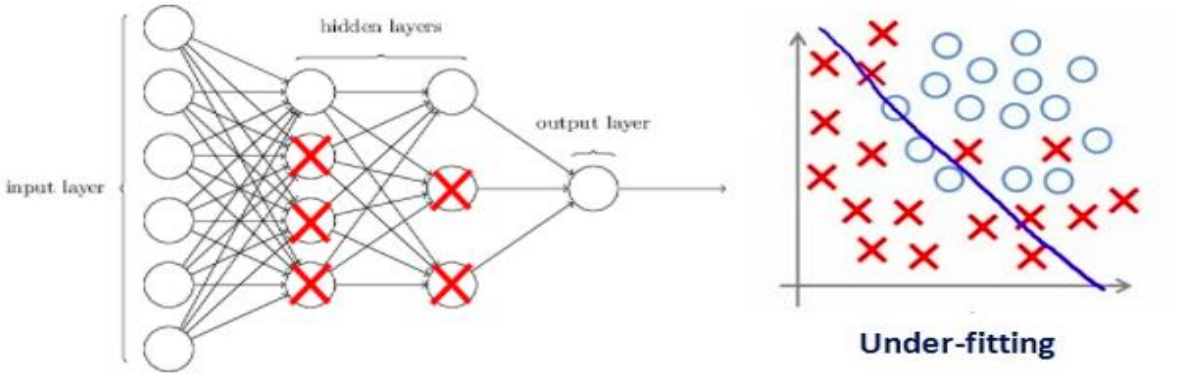


Figure 1.29: Regularization help reduce Under-fitting

This will result in a much simpler linear network and slight underfitting of the training data. Such a large value of the regularization coefficient is not that useful. We need to optimize the value of regularization coefficient in order to obtain a well-fitted model as shown in the image below.

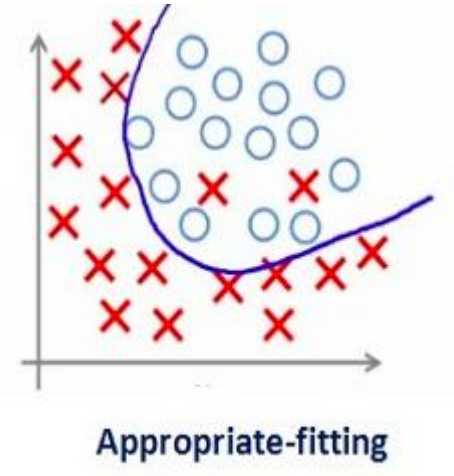


Figure 1.29: Appropriate-fitting

Ridge Regression (L2 Regularization)

- Ridge regression, also known as L2,regularization, is a technique of regularization to avoid the overfitting in training data set, which introduces a small bias in the training model, through which one can get long term predictions for that input.
- In this method, a penalty term is added to the cost function. This amount of bias altered to the cost function in the model is also known as ridge regression penalty hence, the equation for the cost

In L2 Regularization, the term we add to the cost function is the following:

$$\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_2^2$$

In this case, the regularization term is the squared norm of the weights of each network's layer. **This matrix norm is called Frobenius norm and, explicitly, it's computed as follows:**

$$||w^{[l]}||_2^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}(w_{ij}^{[l]})^2$$

Please note that the weight matrix relative to layer l has n^{[l]} rows and n^{[l-1]} columns.
Finally, the complete cost function under L2 Regularization becomes:

$$J(w^{[1]}, b^{[1]}, \cdots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_2^2$$

Again, λ is the regularization term and for λ=0 the effects of L2 Regularization are null.
L2 Regularization brings towards zero the values of the weights, resulting in a simpler model.

- ✓ It regularizes the co-efficient set for the model and hence the ridge regression term deduces the values of the coefficient, which ultimately helps in deducing the complexity of the machine learning model.
- ✓ From the above equation, we can observe that if the value of tends to zero, the last term on the right - hand side will tend to zero,.thus making the above equation a representation of a simple linear regression model:
- ✓ Hence, lower the value of, the model will tend to linear regression.
- ✓ This model is important to execute the neural networks for machine learning, as there would be risks of failure for generalized linear regression models, if there are dependencies found between its variables. Hence, ridge regression is used here.

Lasso Regression (L1 Regularization)

- ✓ One more technique to reduce the overfitting, and thus the complexity of the model is the lasso regression.
- ✓ Lasso regression stands for Least Absolute and Selection Operator and is also sometimes known as L1 regularization.
- ✓ The equation for the lasso regression is almost same as that of the ridge regression, except for a change that the value. of the penalty term is taken as the absolute weights.
- ✓ The advantage of taking the absolute values is that its slope can shrink to 0, as compared to the ridge regression, where the slope will shrink it near to 0.

The following equation gives the cost function defined in the Lasso regression:
In L1 Regularization we add the following term to the cost function J:

$$\frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_1$$

where the matrix norm is the sum of the absolute value of the weights for each layer 1, …, L of the network:

$$||w^{[l]}||_1 = \sum_i \sum_j |w_{ij}^{[l]}|$$

λ is the regularization term. It's a hyperparameter that must be carefully tuned. λ directly controls the impact of the regularization: as λ increases, the effects on the weights shrinking are more severe.
The complete cost function under L1 Regularization becomes:

$$J(w^{[1]}, b^{[1]}, \cdots, w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L}||w^{[l]}||_1$$

For λ=0, the effects of L1 Regularization are null. Instead, choosing a value of λ which is too big, will over-simplify the model, probably resulting in an underfitting network.
L1 Regularization can be considered as a sort of neuron selection process because it would bring to zero the weights of some hidden neurons.

- Due to the acceptance of absolute values for the cost function, some of the features of the input dataset can be ignored completely while evaluating the machine learning model, and hence the feature selection and overfitting can be reduced to much extent.

- On the other hand, ridge regression does not ignore any feature in the model and includes it all for model evaluation. The complexity of the model can be reduced using the shrinking of co-efficient in the ridge regression model.

7 Explain activation functions and their types used in neural networks. 16 K2

Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Types of Activation functions:

- Sign function
- Step function, and
- Sigmoid function

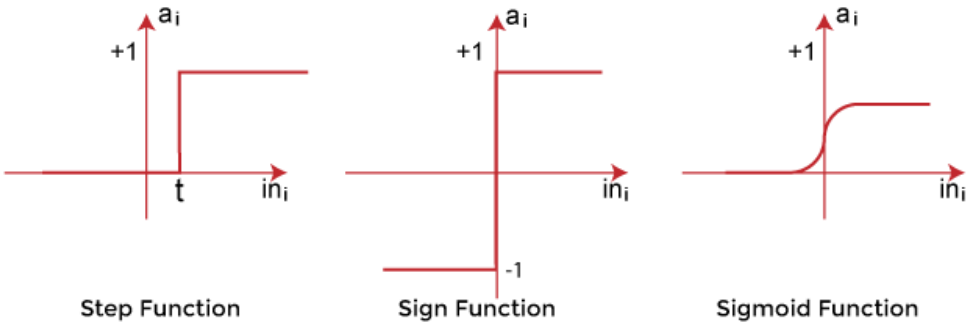


Figure 1.9: Types of Activation functions

The data scientist uses the activation function to take a subjective decision based on various problem statements and forms the desired outputs. Activation function may differ (e.g., Sign, Step, and Sigmoid) in perceptron models by checking whether the learning process is slow or has vanishing or exploding gradients.

How does Perceptron work?

In Machine Learning, Perceptron is considered as a single-layer neural network that consists of four main parameters named input values (Input nodes), weights and Bias, net sum, and an activation function. The perceptron model begins with the multiplication of all input values and their weights, then adds these values together to create the weighted sum. Then this weighted sum is applied to the activation function 'f' to obtain the desired output. This activation function is also known as the step function and is represented by 'f'.

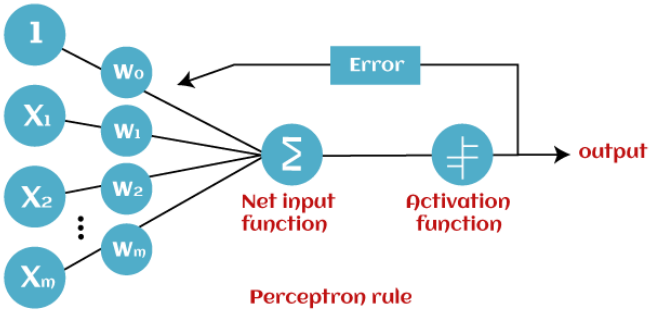


Figure 1.10: Working Process of Perceptron

This step function or Activation function plays a vital role in ensuring that output is mapped between required values (0,1) or (-1,1). It is important to note that the weight of input is indicative of the strength of a node. Similarly, an input's bias value gives the ability to shift the activation function curve up or down.

Perceptron model works in two important steps as follows:

Step-1

In the first step first, multiply all input values with corresponding weight values and then add them to determine the weighted sum. Mathematically, we can calculate the weighted sum as follows:

$$\sum wi * xi = x1 * w1 + x2 * w2 + \cdots wn * xn$$

Add a special term called bias 'b' to this weighted sum to improve the model's performance.

$$\sum wi * xi + b$$

Step-2

In the second step, an activation function is applied with the above-mentioned weighted sum, which gives us output either in binary form or a continuous value as follows:

$$Y = f\left(\sum wi * xi + b\right)$$

8(i) Consider the following five training example **8 K4**

X	Y
2	**8.8978**
3	**11.7586**
4	**15.3192**
5	**18.3129**
6	**20.1351**

We want to learn the function f(x) of the form (x)=ax+b which is parameterized (a,b). Using squared error as the loss function which of the following parameters would you use to model this. function to get a solution with minimum loss? (4,3), (1,4), (4,1) (3,4).

It determine which parameters (a,b) best fit the function f(x)=ax+b with minimum squared error, we need to perform **linear regression** using the given data. The **least squares method** minimizes the sum of squared errors (SSE), given by:

$$SSE = \sum \left(Y_i - (aX_i + b)\right)^2$$

Step 1: Compute Mean of X and Y

Given data points:

X	Y
2	8.8978
3	11.7586
4	15.3192
5	18.3129
6	20.1351

Calculate the means:

$$\bar{X} = \frac{2+3+4+5+6}{5} = \frac{20}{5} = 4$$

$$\bar{Y} = \frac{8.8978 + 11.7586 + 15.3192 + 18.3129 + 20.1351}{5} = \frac{74.4236}{5} = 14.88472$$

Step 2: Compute Slope a

The formula for a (slope) in simple linear regression:

$$a = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$$

Step 3: Compute Intercept b

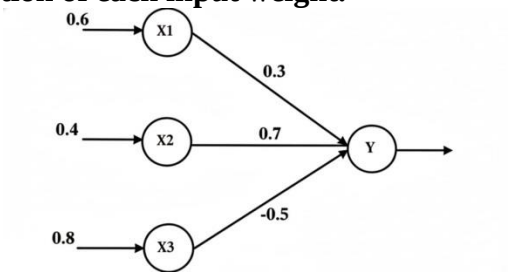
$$b = \bar{Y} - a\bar{X}$$

Now, let's calculate these values to find the best-fitting parameters.

The computed best-fit parameters for f(x)=ax+b are:

- a≈2.9029
- b≈3.2732

8(ii) For the given network shown in figure, analyze and compute the net input to the output neuron Y by considering the contribution of each input weight. **8** **K4**



Let's calculate the output of a perceptron given the inputs and weights:

Inputs: $x1 = 0.6, x2 = 0.4, x3 = 0.8$,

Weight: $w1 = 0.3, w2 = 0.7, w3 = -0.5$

Bias=0

Perceptron Calculation

Step 1: Calculate the Weighted Sum

$$z = w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x3$$

Calculate the weighted sum:

$$\boldsymbol{z = 0.3 \cdot 0.6 + 0.7 \cdot 0.4 + (-0.5) \cdot 0.8}$$

z=0.18+0.28−0.4

z=0.46−0.4

z=0.06