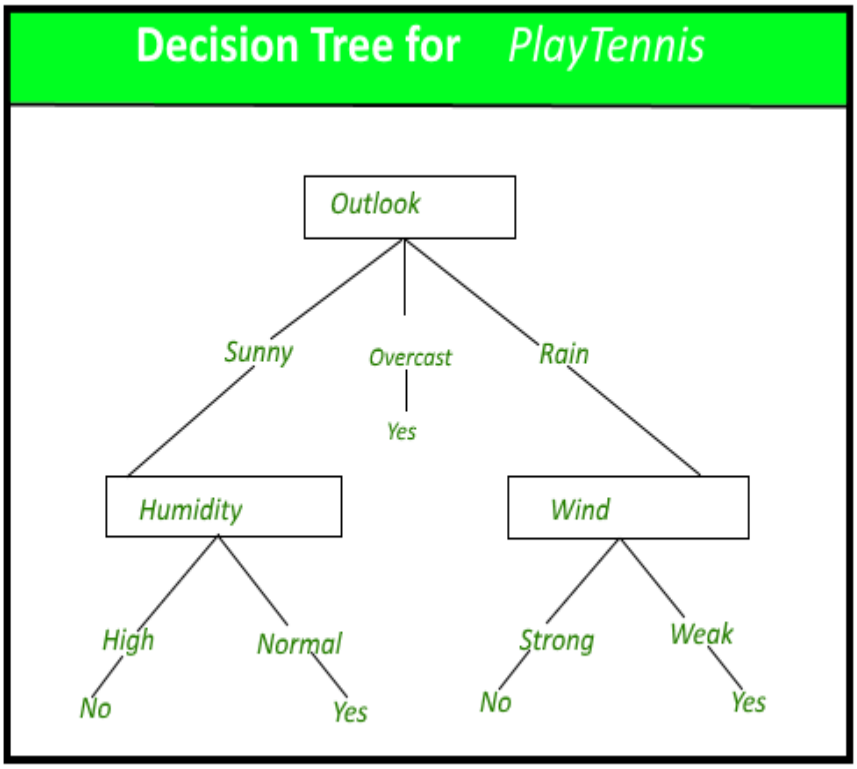
UNIT V

Decision Tree

Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



A decision tree for the concept PlayTennis.

Construction of Decision Tree: A tree can be *"learned"* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of a decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high-dimensional data. In general decision tree, classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Short note on Decision Tree:-

- A decision tree which is also known as prediction tree refers a tree structure to mention the sequences of decisions as well as consequences.
- Considering the input X = (X1, X2,… Xn), the aim is to predict a response or output variable Y.
- Each element in the set (X1, X2,…Xn) is known as input variable.It is possible to achieve the prediction by the process of building a decision tree which has test points as well as branches.
- At each test point, it is decided to select a particular branch and traverse down the tree.
- Ultimately, a final point is reached, and it will be easy to make prediction.
- In a decision tree, all the test points exhibit testing specific input variables (or attributes), and the developed decision tree is represented by the branches.
- Because of flexibility as well as simple visualization, decision trees are mostly probably deployed in data mining applications for the purpose of classification.
- In the decision tree, the input values are considered as categorical or continuous.
- A structure of test points (known as nodes) and branches is established by the decision tree by which the decision being made will be represented.
- Leaf node is the one which do not have further branches. The returning value of leaf nodes is class labels while in some cases they return the probability scores.
- It is possible to convert decision tree into a set of decision rules.
- There are two types of Decision trees: **classification trees and regression trees**
- **Classification trees** are generally applied to output variables which are categorical and mostly binary in nature, for example yes or no, sale or not, and so on.
- Whereas **regression trees** are applied to output variables which are numeric or continuous, for example predicted price of a consumer good.
- In variety of situations, it is possible to apply decision tree. It is easy to represent them in a visual way, and the analogous straightforward.
- Also as the result is a sequence of logical if-then statements, there is no any presence of underlying assumption regarding a linear or nonlinear relationship between the input variables and the response variable.

Decision Tree Representation: Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.

The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returns the classification associated with the particular leaf.(in this case Yes or No).

For example, the instance

(Outlook = Sunny, Temperature = Hot, Humidity = High, Wind = Strong)

would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance.

In other words, we can say that the decision tree represents a disjunction of conjunctions of constraints on the attribute values of instances.

(Outlook = Sunny ^ Humidity = Normal) v (Outlook = Overcast) v (Outlook = Rain ^ Wind = Weak)

Gini Index:

Gini Index is a score that evaluates how accurate a split is among the classified groups. Gini index evaluates a score in the range between 0 and 1, where 0 is when all observations belong to one class, and 1 is a random distribution of the elements within classes. In this case, we want to have a Gini index score as low as possible. Gini Index is the evaluation metrics we shall use to evaluate our Decision Tree Model.

Implementation:

- C++
- Python3
- C#
- Javascript

```
// Importing required headers

#include <algorithm>

#include <chrono>

#include <cmath>

#include <cstdlib>

#include <ctime>

#include <fstream>

#include <iostream>

#include <iterator>

#include <queue>

#include <random>

#include <sstream>

#include <vector>



#include <bits/stdc++.h>

#include <stdio.h>
```

```
using namespace std;


int main()
{
    // Generating random data for classification
    int X[100][5];
    int t[100];
    srand(10);
    for (int i = 0; i < 100; i++) {
        for (int j = 0; j < 5; j++) {
            X[i][j] = rand() % 2;
        }
        t[i] = rand() % 2;
    } // Splitting data into train and test sets
    int X_train[70][5];
    int X_test[30][5];
    int t_train[70];
    int t_test[30];


    for (int i = 0; i < 70; i++) {
        for (int j = 0; j < 5; j++) {
```

```
            X_train[i][j] = X[i][j];
        }
        t_train[i] = t[i];
    }

    for (int i = 0; i < 30; i++) {
        for (int j = 0; j < 5; j++) {
            X_test[i][j] = X[i + 70][j];
        }
        t_test[i] = t[i + 70];
    }

    // Randomly predicting binary values for test set
    int predicted_value[30];
    for (int i = 0; i < 30; i++) {
        predicted_value[i] = rand() % 2;
    }

    // Printing predicted binary values for test set
    for (int i = 0; i < 30; i++) {
        cout << predicted_value[i] << " ";
    }
```

```
cout << "\n";


// Calculating number of 0s and 1s in train set
int zeroes = 0;
int ones = 0;
for (int i = 0; i < 70; i++) {
    if (t_train[i] == 0) {
        zeroes += 1;
    }
    else {
        ones += 1;
    }
}


// Calculating Gini index
float val = 1
            - ((zeroes / 70.0) * (zeroes / 70.0)
               + (ones / 70.0) * (ones / 70.0));
cout << "Gini : " << val << "\n";


// Calculating accuracy of predictions
int match = 0;
```

```
    int UnMatch = 0;

    for (int i = 0; i < 30; i++) {

        if (predicted_value[i] == t_test[i]) {

            match += 1;

        }

        else {

            UnMatch += 1;

        }

    }

    float accuracy = match / 30.0;

    cout << "Accuracy is: " << accuracy << "\n";



    // Returning 0 on successful completion

    return 0;

}
```

Output

```
1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0

Gini : 0.5

Accuracy is: 0.366667
```

Strengths and Weaknesses of the Decision Tree approach

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

- Ease of use: Decision trees are simple to use and don't require a lot of technical expertise, making them accessible to a wide range of users.
- Scalability: Decision trees can handle large datasets and can be easily parallelized to improve processing time.
- Missing value tolerance: Decision trees are able to handle missing values in the data, making them a suitable choice for datasets with missing or incomplete data.
- Handling non-linear relationships: Decision trees can handle non-linear relationships between variables, making them a suitable choice for complex datasets.
- Ability to handle imbalanced data: Decision trees can handle imbalanced datasets, where one class is heavily represented compared to the others, by weighting the importance of individual nodes based on the class distribution.

The weaknesses of decision tree methods :

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many classes and a relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

Steps for making decision tree

Follow these five steps to create a decision tree diagram to analyze uncertain outcomes and reach the most logical solution.

1. Start with your idea. Begin your diagram with one main idea or decision. ...
2. Add chance and decision nodes. ...
3. Expand until you reach end points. ...
4. Calculate tree values. ...
5. Evaluate outcomes.
6.

Start with your idea

Begin your diagram with one main idea or decision. You'll start your tree with a decision node before adding single branches to the various decisions you're deciding between.

For example, if you want to create an app but can't decide whether to build a new one or upgrade an existing one, use a decision tree to assess the possible outcomes of each.

In this case, the initial decision node is:

- Create an app

The three options—or branches—you're deciding between are:

- Building a new scheduling app

- Upgrading an existing scheduling app
- Building a team productivity app

2. Add chance and decision nodes

After adding your main idea to the tree, continue adding chance or decision nodes after each decision to expand your tree further. A chance node may need an alternative branch after it because there could be more than one potential outcome for choosing that decision.

For example, if you decide to build a new scheduling app, there's a chance that your revenue from the app will be large if it's successful with customers. There's also a chance the app will be unsuccessful, which could result in a small revenue. Mapping both potential outcomes in your decision tree is key.

3. Expand until you reach end points

Keep adding chance and decision nodes to your decision tree until you can't expand the tree further. At this point, add end nodes to your tree to signify the completion of the tree creation process.

Once you've completed your tree, you can begin analyzing each of the decisions.

4. Calculate tree values

Ideally, your decision tree will have quantitative data associated with it. The most common data used in decision trees is monetary value.

For example, it'll cost your company a specific amount of money to build or upgrade an app. It'll also cost more or less money to create one app over another. Writing these values in your tree under each decision can help you in the decision-making process.

You can also try to estimate expected value you'll create, whether large or small, for each decision. Once you know the cost of each outcome and the probability it will occur, you can calculate the expected value of each outcome using the following formula:

- Expected value (EV) = (First possible outcome x Likelihood of outcome) + (Second possible outcome x Likelihood of outcome) - Cost

Calculate the expected value by multiplying both possible outcomes by the likelihood that each outcome will occur and then adding those values. You'll also need to subtract any initial costs from your total.

5. Evaluate outcomes

Once you have your expected outcomes for each decision, determine which decision is best for you based on the amount of risk you're willing to take. The highest expected value may not always be the one you want to go for. That's because, even though it could result in a high reward, it also means taking on the highest level of project risk.

Keep in mind that the expected value in decision tree analysis comes from a probability algorithm. It's up to you and your team to determine how to best evaluate the outcomes of the tree.

What is Feed-Forward Neural Networks?

Feed-forward neural networks allows signals to travel one approach only, from input to output. There is no feedback (loops) such as the output of some layer does not influence that same layer. Feed-forward networks tends to be simple networks that associates inputs with outputs. It can be used in pattern recognition. This type of organization is represented as bottom-up or top-down.

Each unit in the hidden layer is generally completely connected to some units in the input layer. Because this network includes standard units, the units in the hidden layer compute their output by multiplying the value of each input by its correlating weight, inserting these up, and using the transfer function.

A neural network can have several hidden layers, but as usual, one hidden layer is adequate. The wider the layer the higher the capacity of the network to identify designs.

The final unit on the right is the output layer because it is linked to the output of the neural network. It is completely connected to some units in the hidden layer. The neural network is generally used to compute a single value, therefore there is only one unit in the output layer and the value.

It is applicable for the output layer to have higher than one unit. For example, a department store chain required to forecast the likelihood that users will be buying products from several departments, including women's apparel, furniture, and entertainment. The stores required to need this data to plan promotions and direct focus mailings.

The backpropagation algorithm performs learning on a multilayer feed-forward neural network. The inputs stimulate the attributes computed for each training sample. The inputs are fed into a layer of units making up the input layer.

The weighted outputs of these units are fed concurrently to the second layer of neurons like units called the hidden layer. The hidden layer is weighted output which can be input to multiple hidden layer etc. The multiple hidden layers is arbitrary and frequently, one is used.

The weighted outputs of the final hidden layer are inputs to units creating up the output layer, which diffuse the network's prediction for provided samples. The units in the hidden layers and output layer are represented as neurodes, due to their symbolic biological elements or as output units. Multilayer feed-forward networks of linear threshold functions provided through hidden units can nearly approximate some function.

What Is a Neural Network?

A neural network is a collection of neurons that take input and, in conjunction with information from other nodes, develop output without programmed rules. Essentially, they solve problems through trial and error.

Neural networks are based on human and animal brains. While neural networks are advanced enough to beat human opponents at gamesExternal link:open_in_new like chess and Go, they lack the cognitive abilities of a human toddler and most animals.

Neural Network Elements

A neural network is made up of densely connected processing nodes, similar to neurons in the brain. Each node may be connected to different nodes in multiple layers above and below it. These nodes move data through the network in a feed-forward fashion, meaning the data moves in only one direction. The node “fires” like a neuron when it passes information to the next node.

A simple neural network has an input layer, output layer and one hidden layer between them. A network with more than three layers, including the input and output, is known as a deep learning network. In a deep learning network, each layer of nodes trains on data based on the output from the previous layer. The more layers, the greater the ability to recognize more complex information — based on data from the previous layers.

The network makes decisions by assigning each connected node to a number known as a “weight.” The weight represents the value of information assigned to an individual node (i.e., how helpful it is in correctly classifying information). When a node receives information from other nodes, it calculates the total weight or value of the information. If the number exceeds a certain threshold, the information is passed onto the next layer. If the weight is below the threshold, the information is not passed on.

In a newly formed neural network, all weights and thresholds are set to random numbers. As training data is fed into the input layer, the weights and thresholds refine to consistently yield correct outputs.

How Does a Neural Network Work?

Whether it’s biological or artificial, the power of a neural network stems from the way simple neurons are linked to form a complex system greater than the sum of its parts.

Each neuron can make simple decisions based on mathematical calculations. Together, many neurons can analyze complex problems and provide accurate answers. A shallow network is composed of an input, hidden layer and output layer. A deep neural network has more than one hidden layer, which increases the complexity of the problems it can analyze.

A neural network learns to complete a task by examining labeled training examples. The samples must be labeled so the network can learn to distinguish between items using visual patterns correlated with the labels.

A neural network has three functions:

- Scoring input
- Calculating loss
- Updating the model, which begins the process over again

A neural network is a corrective feedback loop, giving more weight to data that supports correct guesses and less weight to data that leads to mistakes. A feature known as backpropagation trains the network to identify correct responses and ignore incorrect responses.

Advantages of Neural Networks

As we have now understood the basics of neural networks and the way they work, let us now dig into the advantages of neural networks.

1. **Effective Visual Analysis**

The very first advantage of neural networks is that they lead to an effective visual analysis. Since an artificial neural network is similar to that of a human's neural network, it is capable of performing more complex tasks and activities as compared to other machines.

This involves analyzing visual information and segregating it into different categories. A typical example of this advantage is when any website that you visit asks you to prove whether or not you are a robot.

Robots cannot effectively analyze visual information, while humans can successfully do so. This proves that any person logging into a website is a human as s/he is required to differentiate between different images and put images of a certain kind together.

2. **Processing of Unorganized Data**

Another one of the greatest advantages of neural networks is that it is capable of processing unorganized data. Have you ever wondered how artificial intelligence and machine learning organize bits of data?

The answer lies in the ability of neural networks. By processing, segregating, and categorizing unorganized data, artificial neural networks or ANNs can very well organize data.

In collaboration with big data analytics, unorganized data can be structured into a similar pattern and in turn, organized. With the coming of ANNs, the task of organizing

unorganized data has particularly gotten a lot easier.

Unlike the traditional times when teams of skilled humans had to invest their days in categorizing unorganized data, today computers can perform the same function in a span of minutes, if not seconds.

3. **Adaptive Structure**

The third advantage of neural networks is that their structure is adaptive in nature. This means that for whatever purpose an ANN is applied, it alters its course of the structure according to the purpose.

From developing the cognitive abilities of a machine to performing complex applications, the structure of the neural networks is subject to change. This is as opposed to the otherwise fairly rigid structures of numerous machine learning algorithms and applications.

Unlike unchangeable structures, artificial neural networks quickly transform, adapt, and adjust to new environments and display their skills accordingly. This also indicates that the kind of training that goes into the training of these networks is comparatively lesser and more accommodative.

4. **User-friendly Interface**

The last advantage among others is that they portray a user-friendly interface. For any machine or artificial equipment to become a success, its interface and usability of it should be user-friendly.

Likewise, it should not be too complex to work with and simple to use throughout. This goes for describing artificial neural networks in the best possible way. With a user-friendly interface, ANNs can be trained without too many complexities.

Likewise, they are capable of adapting their structures which makes them even more helpful for semi-skilled and skilled professionals to operate. This is one of the biggest advantages of this concept as it can easily adjust with any team of professionals aiming to work with it.

There are a variety of varying networks that function independently in the neural network and perform many sub-tasks. There is no requirement for any sort of interaction amongst each other during the computation process.

Disadvantages of Neural Networks

Even though the benefits of neural networks outnumber their disadvantages, it is important to consider them and even dig deep into their whereabouts. Let us now read about some of the well-known disadvantages of neural networks.

1. **Hardware Requirement**

Despite their ability to quickly adapt to the changing requirements of the purpose they are supposed to work for, neural networks can be a bit hefty to arrange and organize. This means that they require heavy machinery and hardware equipment to work for any application.

For beginners or those on a tight budget, this might be one of the obstacles of neural networks. Moreover, it can also mean that one has to invest in supplementary things more than the main component of the process.

Thus, artificial neural networks can be a bit problematic when it comes to their hardware setting, organization, and placement.

2. **Incomplete Results**

The second demerit of neural networks is that they can often create incomplete results or outputs. Since ANNs are trained to adapt to the changing applications of neural networks, they are often left untrained for the whole process.

While this seems to be a fairly easy aspect when it comes to the benefits of ANNs, it can quickly turn into a disadvantage as soon as it is time for the output. Due to incomplete results, ANNs have many a time been the talk of the town. With the help of numerous theorems, only a probable value or an estimate can be calculated for such networks.

3. **Data Suitability**

Another one of the challenges of neural networks is that they are highly dependent on the data made available to them. This infers that the efficiency of any neural network is directly proportional to the amount of data it receives to process.

What's more, ANNs are also affected if the data made available to them is not suitable enough. Thus, artificial neural network algorithms can go wrong while analyzing data available in small amounts and the one that they cannot interpret easily.

Even when these networks are being trained, they should be fed with humongous data to prepare them for the future. If not, then the results can possibly turn out to be faulty and can distort the actual findings of computation, application, or simply a task.

4. **Minimal Control**

While artificial neural networks programs are pretty much advantageous when it comes to organizing unorganized data, they can be highly damaging too. This refers to the minimal control that the trainers have over the actual performance and overall functioning of the ANNs.

From probable value to the unknown steps of working, artificial neural networks are pretty much concealed in their actual structure. This can mean that not much external influence or control can be exerted on these networks to run them as per the user's convenience.

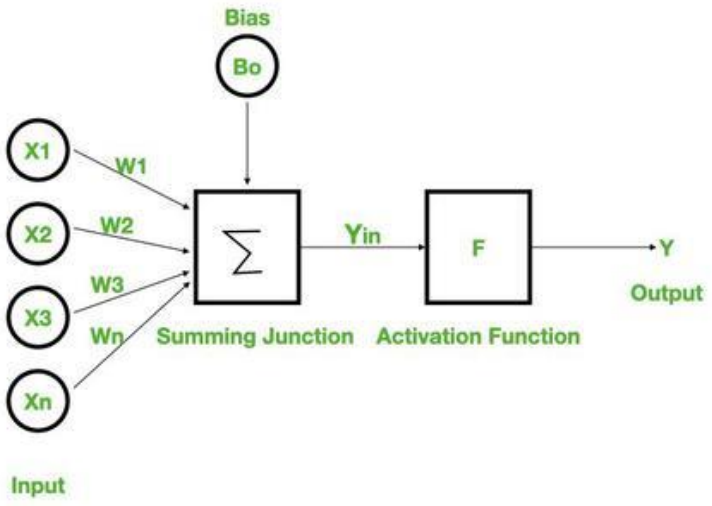
Backpropagation

Backpropagation is an algorithm that backpropagates the errors from the output nodes to the input nodes. Therefore, it is simply referred to as the backward propagation of errors. It uses in the vast applications of neural networks in data mining like Character recognition, Signature verification, etc.

Neural Network:

Neural networks are an information processing paradigm inspired by the human nervous system. Just like in the human nervous system, we have biological neurons in the same way in neural networks we

have artificial neurons, artificial neurons are mathematical functions derived from biological neurons. The human brain is estimated to have about 10 billion neurons, each connected to an average of 10,000 other neurons. Each neuron receives a signal through a synapse, which controls the effect of the signconcerning on the neuron.



Backpropagation:

Backpropagation is a widely used algorithm for training feedforward neural networks. It computes the gradient of the loss function with respect to the network weights. It is very efficient, rather than naively directly computing the gradient concerning each weight. This efficiency makes it possible to use gradient methods to train multi-layer networks and update weights to minimize loss; variants such as gradient descent or stochastic gradient descent are often used.

The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight via the chain rule, computing the gradient layer by layer, and iterating backward from the last layer to avoid redundant computation of intermediate terms in the chain rule.

Features of Backpropagation:

1. it is the gradient descent method as used in the case of simple perceptron network with the differentiable unit.

2. it is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
3. training is done in the three stages :
   - the feed-forward of input training pattern
   - the calculation and backpropagation of the error
   - updation of the weight

Working of Backpropagation:

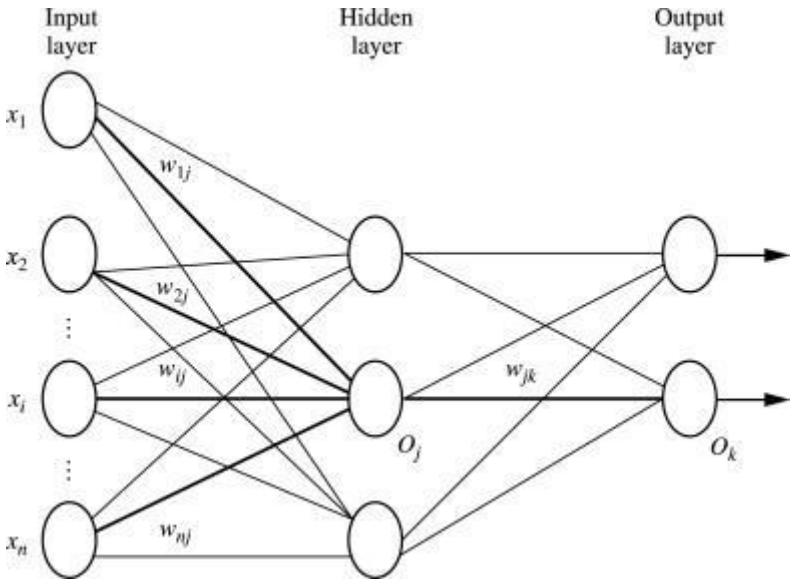
Neural networks use supervised learning to generate output vectors from input vectors that the network operates on. It Compares generated output to the desired output and generates an error report if the result does not match the generated output vector. Then it adjusts the weights according to the bug report to get your desired output.

Backpropagation Algorithm:

Step 1: Inputs X, arrive through the preconnected path.
Step 2: The input is modeled using true weights W. Weights are usually chosen randomly.
Step 3: Calculate the output of each neuron from the input layer to the hidden layer to the output layer.
Step 4: Calculate the error in the outputs

```
Backpropagation Error= Actual Output – Desired Output
```

Step 5: From the output layer, go back to the hidden layer to adjust the weights to reduce the error.
Step 6: Repeat the process until the desired output is achieved.



Parameters :

- x = inputs training vector $x=(x_1,x_2,\ldots\ldots\ldots\ldots x_n)$.
- t = target vector $t=(t_1,t_2\ldots\ldots\ldots\ldots t_n)$.
- δ_k = error at output unit.
- δ_j = error at hidden layer.

- α = learning rate.
- V_{0j} = bias of hidden unit j.

Training Algorithm :

Step 1: Initialize weight to small random values.

Step 2: While the stepsstopping condition is to be false do step 3 to 10.

Step 3: For each training pair do step 4 to 9 (Feed-Forward).

Step 4: Each input unit receives the signal unit and transmitsthe signal x_i signal to all the units.

Step 5 : Each hidden unit Zj (z=1 to a) sums its weighted input signal to calculate its net input

$$z_{inj} = v_{0j} + \Sigma x_i v_{ij} \quad (i=1 \text{ to } n)$$

Applying activation function $z_j = f(z_{inj})$ and sends this signals to all units in the layer about i.e output units

For each output l=unit y_k = (k=1 to m) sums its weighted input signals.

$$y_{ink} = w_{0k} + \Sigma z_i w_{jk} \quad (j=1 \text{ to } a)$$

and applies its activation function to calculate the output signals.

$$y_k = f(y_{ink})$$

Backpropagation Error :

Step 6: Each output unit y_k (k=1 to n) receives a target pattern corresponding to an input pattern then error is calculated as:

$$\delta_k = (t_k - y_k) + y_{ink}$$

Step 7: Each hidden unit Z_j (j=1 to a) sums its input from all units in the layer above

$$\delta_{inj} = \Sigma \delta_j w_{jk}$$

The error information term is calculated as :

$$\delta_j = \delta_{inj} + z_{inj}$$

Updation of weight and bias :

Step 8: Each output unit y_k (k=1 to m) updates its bias and weight (j=1 to a). The weight correction term is given by :

$$\Delta w_{jk} = \alpha \delta_k z_j$$

and the bias correction term is given by $\Delta w_k = \alpha \delta_k$.

therefore $w_{jk(new)} = w_{jk(old)} + \Delta w_{jk}$

$$w_{0k(new)} = w_{ok(old)} + \Delta w_{ok}$$

for each hidden unit z_j (j=1 to a) update its bias and weights (i=0 to n) the weight connection term

$$\Delta v_{ij} = \alpha \delta_j x_i$$

and the bias connection on term

$$\Delta v_{0j} = \alpha \delta_j$$

Therefore $v_{ij(new)} = v_{ij(old)} + \Delta v_{ij}$

$$v_{0j(new)} = v_{0j(old)} + \Delta v_{0j}$$

Step 9: Test the stopping condition. The stopping condition can be the minimization of error, number of epochs.

Need for Backpropagation:

Backpropagation is “backpropagation of errors” and is very useful for training neural networks. It’s fast, easy to implement, and simple. Backpropagation does not require any parameters to be set, except the number of inputs. Backpropagation is a flexible method because no prior knowledge of the network is

required.

Types of Backpropagation

There are two types of backpropagation networks.

- **Static backpropagation:** Static backpropagation is a network designed to map static inputs for static outputs. These types of networks are capable of solving static classification problems such as OCR (Optical Character Recognition).
- **Recurrent backpropagation:** Recursive backpropagation is another network used for fixed-point learning. Activation in recurrent backpropagation is feed-forward until a fixed value is reached. Static backpropagation provides an instant mapping, while recurrent backpropagation does not provide an instant mapping.

Advantages:

- It is simple, fast, and easy to program.
- Only numbers of the input are tuned, not any other parameter.
- It is Flexible and efficient.
- No need for users to learn any special functions.

Disadvantages:

- It is sensitive to noisy data and irregularities. Noisy data can lead to inaccurate results.
- Performance is highly dependent on input data.
- Spending too much time training.
- The matrix-based approach is preferred over a mini-batch.