SCT UNIT-2 - SCT

COMPUTER SCIENCE ENGINEERING (Jawaharlal Nehru Technological University, Kakinada)

SOFT COMPUTING TECHNIQUES

UNIT-2

UNIT -II:

Artificial Neural Networks: Introduction to Artificial Neural Networks, Classification of ANNS, First generation neural networks, Perceptron network, Adaline, Madaline, Second generation neural networks, Back propagation neural networks, Hopfield Neural Network, Kohonen neural network, Hamming neural network, Radial basis function neural networks, spike neuron models.

Classification of ANNS

Artificial Neural Networks (ANNs) can be classified based on various criteria, including architecture, learning algorithm, and application. Here are some common classifications:

1. Based on Architecture:

- Single-Layer Perceptron (SLP): Consists of a single layer of input nodes connected directly to the output nodes. It's suitable for linearly separable problems.
- Multi-Layer Perceptron (MLP): Consists of an input layer, one or more hidden layers, and an output layer. It can learn complex, non-linear mappings.
- Feedforward Neural Network (FNN): Information travels in one direction—from input to output—without loops or cycles.
- Recurrent Neural Network (RNN): Contains cycles or loops in the network, allowing information to persist. Suitable for sequence-based data.

2. Based on Learning Algorithm:

- Supervised Learning: The network is trained on a labeled dataset where the input-output pairs are provided.
- Unsupervised Learning: The network learns patterns and relationships in the data without explicit supervision.
- Semi-Supervised Learning: A combination of labeled and unlabeled data is used for training.
- Reinforcement Learning: The network learns through trial and error by interacting with an environment and receiving feedback in the form of rewards or penalties.

3. Based on Training Approach:

- Batch Learning: The network is trained on the entire dataset in one go.
- Online Learning (Stochastic Gradient Descent): The network is updated after each individual data point.
- Mini-Batch Learning: The network is updated after processing a small batch of data.

4. Based on Activation Function:

- Sigmoidal (Logistic) Activation: Commonly used in the output layer for binary classification problems.
- Hyperbolic Tangent (tanh) Activation: Similar to sigmoid but with a range of [-1, 1].
- Rectified Linear Unit (ReLU): Often used in hidden layers for its simplicity and effectiveness.
- Softmax Activation: Used in the output layer for multi-class classification problems.

5. Based on Application:

- Pattern Recognition: ANNs are used for image and speech recognition, character recognition, etc.
- Time Series Prediction: RNNs are particularly effective for predicting sequential data.
- Natural Language Processing (NLP): ANNs are applied to tasks like sentiment analysis, machine translation, and text generation.
- Computer Vision: ANNs are used for tasks like object detection, image segmentation, and facial recognition.

6. Based on Topology:

- Fully Connected Networks: Each neuron is connected to every neuron in the adjacent layers.
- Convolutional Neural Networks (CNN): Specialized for processing grid-structured data, like images.
- Radial Basis Function Networks (RBFN): Uses radial basis functions as activation functions.
- Autoencoders: Learn efficient representations of data by encoding and decoding it.

These classifications are not mutually exclusive, and hybrid architectures often combine elements from different categories to address specific challenges in various applications.

First generation neural networks

When referring to "first-generation neural networks," it typically pertains to the early developments in artificial neural networks (ANNs). The evolution of neural networks can be broadly categorized into different generations based on the chronological development and advancements in their architecture, training algorithms, and applications.

First-generation neural networks include the initial models and ideas that laid the foundation for later developments. Here are some key aspects:

1. McCulloch-Pitts Neuron Model (1943): The concept of artificial neurons was introduced by Warren McCulloch and Walter Pitts. They proposed a simplified mathematical model of a neuron, which served as the basis for later developments in neural networks.
2. Perceptron (1957): Frank Rosenblatt developed the perceptron, which is a single-layer neural network. It was designed for binary classification and had a simple learning algorithm. However, it was limited in its ability to handle more complex problems.

3. Limitations and the Perceptron Convergence Theorem (1969): The perceptron's limitations were highlighted by Marvin Minsky and Seymour Papert in their book "Perceptron." They showed that a single-layer perceptron had limitations in solving problems that were not linearly separable.
4. Decline in Interest (1970s-1980s): Due to the perceived limitations of early neural network models, interest in neural networks waned, and the field experienced a period of reduced attention.

It's important to note that first-generation neural networks were primarily focused on single-layer architectures, and their limitations became apparent when faced with more complex, non-linear problems. The subsequent generations of neural networks, including multi-layer perceptron and the development of backpropagation as a training algorithm, addressed some of these limitations and led to the resurgence of interest in neural networks in the late 1980s and 1990s. The advancements in the second and third generations paved the way for the deep learning revolution that we see today.

Perceptron network

Perceptron is one of the simplest Artificial neural network architectures. It was introduced by Frank Rosenblatt in 1957s. It is the simplest type of feedforward neural network, consisting of a single layer of input nodes that are fully connected to a layer of output nodes. It can learn the linearly separable patterns. it uses slightly different types of artificial neurons known as threshold logic units (TLU). it was first introduced by McCulloch and Walter Pitts in the 1940s.

Types of Perceptron

- **Single-Layer Perceptron:** This type of perceptron is limited to learning linearly separable patterns. effective for tasks where the data can be divided into distinct categories through a straight line.
- **Multilayer Perceptron:** Multilayer perceptron possess enhanced processing capabilities as they consist of two or more layers, adept at handling more complex patterns and relationships within the data.

Basic Components of Perceptron

A perceptron, the basic unit of a neural network, comprises essential components that collaborate in information processing.

- Input Features: The perceptron takes multiple input features, each input feature represents a characteristic or attribute of the input data.
- Weights: Each input feature is associated with a weight, determining the significance of each input feature in influencing the perceptron’s output. During training, these weights are adjusted to learn the optimal values.
- Summation Function: The perceptron calculates the weighted sum of its inputs using the summation function. The summation function combines the inputs with their respective weights to produce a weighted sum.

- Activation Function: The weighted sum is then passed through an activation function. Perceptron uses Heaviside step function functions. which take the summed values as input and compare with the threshold and provide the output as 0 or 1.
- Output: The final output of the perceptron, is determined by the activation function's result. For example, in binary classification problems, the output might represent a predicted class (0 or 1).
- Bias: A bias term is often included in the perceptron model. The bias allows the model to make adjustments that are independent of the input. It is an additional parameter that is learned during training.
- Learning Algorithm (Weight Update Rule): During training, the perceptron learns by adjusting its weights and bias based on a learning algorithm. A common approach is the perceptron learning algorithm, which updates weights based on the difference between the predicted output and the true output.

These components work together to enable a perceptron to learn and make predictions. While a single perceptron can perform binary classification, more complex tasks require the use of multiple perceptron organized into layers, forming a neural network.

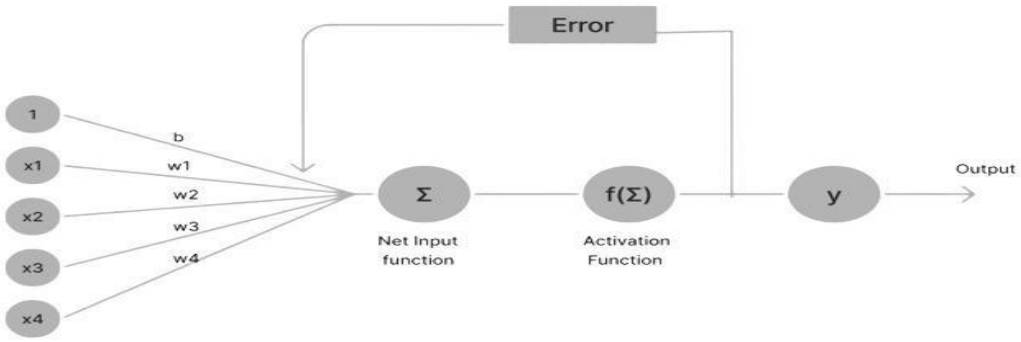
Limitations of Perceptron

- Limited to linearly separable problems.
- Convergence issues with non-separable data
- Requires labeled data
- Sensitivity to input scaling
- Lack of hidden layers

Adaline

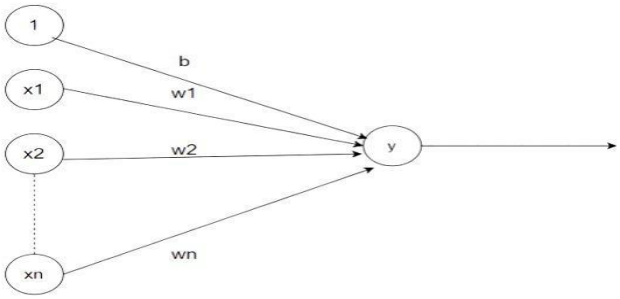
- A network with a single linear unit is called Adaline (Adaptive Linear Neural). A unit with a linear activation function is called a linear unit. In Adaline, there is only one output unit and output values are bipolar (+1,-1). Weights between the input unit and output unit are adjustable. It uses the delta rule i.e w_{i}(new)=w_{i}(old) + α(t-y_{in})x_{i} , where w_{i} y_{in} and t are the weight, predicted output, and true value respectively.
- The learning rule is found to minimize the mean square error between activation and target values. Adaline consists of trainable weights, it compares actual output with calculated output, and based on error training algorithm is applied.

Workflow:



First, calculate the net input to your Adaline network then apply the activation function to its output then compare it with the original output if both the equal, then give the output else send an error back to the network and update the weight according to the error which is calculated by the delta learning rule. i.e w_{i}(new)=w_{i}(old) + α(t-y_{in})x_{i} , where w_{i} y_{in} and t are the weight, predicted output, and true value respectively.

Architecture:



In Adaline, all the input neuron is directly connected to the output neuron with the weighted connected path. There is a bias b of activation function 1 is present.

Algorithm:

Step 1: Initialize weight not zero but small random values are used. Set learning rate α.

Step 2: While the stopping condition is False do steps 3 to 7.

Step 3: for each training set perform steps 4 to 6.

Step 4: Set activation of input unit xi = si for (i=1 to n).

Step 5: compute net input to output unit

$$y_{in} = \sum w_ix_i + b$$

Here, b is the bias and n is the total number of neurons.

Step 6: Update the weights and bias for i=1 to n

$$w_{i}(new) = w_{i}(old) + \alpha(t-y_{in})x_{i} \\ b(new)=b(old) + α(t-y_{in})$$

and calculate

error : (t-y_{in})^2

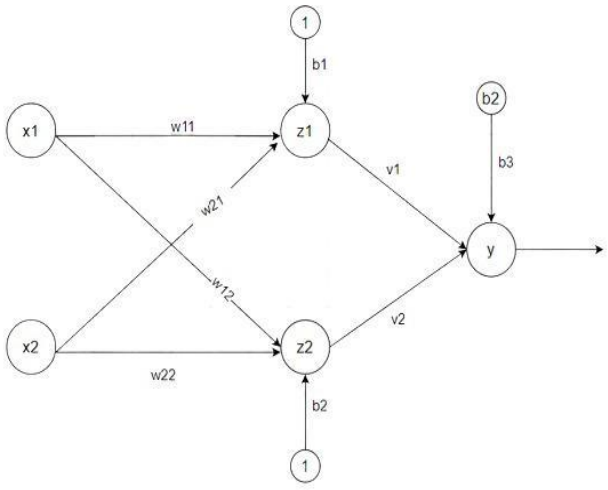
when the predicted output and the true value are the same then the weight will not change.

Step 7: Test the stopping condition. The stopping condition may be when the weight changes at a low rate or no change.

Madaline

- The Madaline (supervised Learning) model consists of many Adaline in parallel with a single output unit. The Adaline layer is present between the input layer and the Madaline layer hence Adaline layer is a hidden layer. The weights between the input layer and the hidden layer are adjusted, and the weight between the hidden layer and the output layer is fixed.
- It may use the majority vote rule, the output would have an answer either true or false. Adaline and Madaline layer neurons have a bias of '1' connected to them. use of multiple Adaline helps counter the problem of non-linear separability.

Architecture:



There are three types of a layer present in Madaline First input layer contains all the input neurons, the Second hidden layer consists of an adaline layer, and weights between the input and hidden layers are adjustable and the third layer is the output layer the weights between hidden and output layer is fixed they are not adjustable.

Algorithm:

Step 1: Initialize weight and set learning rate α.

v1=v2=0.5 , b=0.5

other weight may be a small random value.

Step 2: While the stopping condition is False do steps 3 to 9.

Step 3: for each training set perform steps 4 to 8.

Step 4: Set activation of input unit xi = si for (i=1 to n).

Step 5: compute net input of Adaline unit

$$zin1 = b1 + x1w11 + x2w21$$

$$zin2 = b2 + x1w12 + x2w22$$

Step 6: for output of remote Adaline unit using activation function given below:

$$\text{Activation function } f(z) = 1\ \text{if}\ z \geq 0\ \text{(and)}\ (-1)\ \text{if}\ z<0 \quad .$$

$$z1=f(zin1)$$

$$z2=f(zin2)$$

Step 7: Calculate the net input to output.

$$yin = b3 + z1v1 + z2v2$$

Apply activation to get the output of the net

$$y=f(yin)$$

Step 8: Find the error and do weight updation

if $t \neq y$ then t=1 update weight on z(j) unit whose next input is close to 0.

if t = y no updation

$$wij(new) = wij(old) + \alpha(t-zinj)xi$$

$$bj(new) = bj(old) + \alpha(t-zinj)$$

if t=-1 then update weights on all unit zk which have positive net input

Step 9: Test the stopping condition; weights change all number of epochs.

Second Generation Neural Network

The second generation of neural networks represents a period of advancements and innovations that addressed some of the limitations of the first generation. Key developments during this period include the introduction of multi-layer perceptrons (MLPs), the development of new training algorithms, and increased interest in neural network research. Here are some key aspects of the second generation of neural networks:

1. Multi-Layer Perceptrons (MLPs):

- MLPs introduced multiple hidden layers between the input and output layers, allowing for the modeling of more complex, non-linear relationships in data.

- The addition of hidden layers enabled the networks to learn hierarchical representations of data, overcoming the limitations of single-layer perceptron.

2. Backpropagation Algorithm:

- The backpropagation algorithm, initially proposed in the 1970s but popularized in the 1980s, became a fundamental method for training multi-layer neural networks.
- Backpropagation uses gradient descent to update the weights of the network, reducing the error between predicted and actual outputs.

3. Widening Applications:

- Neural networks gained popularity in various applications, including pattern recognition, speech recognition, image processing, and financial modeling.
- The increased computational power and availability of data contributed to the broader adoption of neural networks.

4. Introduction of Activation Functions:

- The use of non-linear activation functions, such as the sigmoid and hyperbolic tangent functions, became common in hidden layers. These functions allowed networks to model non-linear mappings.

5. Development of Convolutional Neural Networks (CNNs):

- CNNs, a specialized type of neural network designed for image and visual data processing, were introduced during this period.
- CNNs use convolutional layers to automatically and adaptively learn spatial hierarchies of features in data.

6. Time-Delay Neural Networks (TDNN):

- TDNNs, designed for sequence modeling and prediction, were developed as an extension of MLPs to handle temporal dependencies in data.

7. Hybrid Systems:

- Researchers explored the combination of neural networks with other machine learning techniques, leading to the development of hybrid systems for improved performance.

The second generation of neural networks laid the groundwork for the later emergence of deep learning. While progress was made, challenges such as vanishing gradients and limited computational resources still hindered the training of very deep networks during this era. The third generation, characterized by the rise of deep learning and the development of more advanced architectures, further addressed these challenges.

Back propagation neural networks

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.

Hopfield Neural Network

The Hopfield Neural Networks, invented by Dr John J. Hopfield consists of one layer of ‘n’ fully connected recurrent neurons. It is generally used in performing auto-association and optimization tasks. It is calculated using a converging interactive process and it generates a different response than our normal neural nets.

Discrete Hopfield Network

It is a fully interconnected neural network where each unit is connected to every other unit. It behaves in a discrete manner, i.e. it gives finite distinct output, generally of two types:

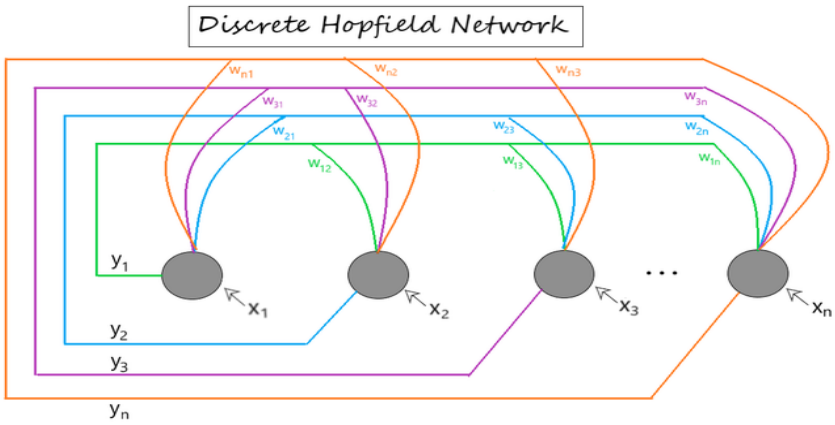
- Binary (0/1)
- Bipolar (-1/1)

The weights associated with this network are symmetric in nature and have the following properties.

Structure & Architecture of Hopfield Network

- Each neuron has an inverting and a non-inverting output.
- Being fully connected, the output of each neuron is an input to all other neurons but not the self.

The below figure shows a sample representation of a Discrete Hopfield Neural Network architecture having the following elements.



Continuous Hopfield Network

Unlike the discrete Hopfield networks, here the time parameter is treated as a continuous variable. So, instead of getting binary/bipolar outputs, we can obtain values that lie between 0 and 1. It can be used to solve constrained optimization and associative memory problems. The output is defined as: v_i = g(u_i)

Energy Function

The Hopfield networks have an energy function associated with them. It either diminishes or remains unchanged on update (feedback) after every iteration. The energy function for a continuous Hopfield network is defined as:

E = 0.5 \sum_{i=1}^n \sum_{j=1}^n w_{ij}v_i v_j + \sum_{i=1}^n \theta_i v_i

To determine if the network will converge to a stable configuration, we see if the energy function reaches its minimum by:

\frac{d}{dt} E \leq 0

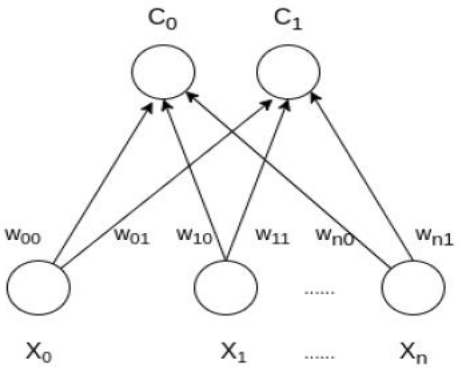
The network is bound to converge if the activity of each neuron wrt time is given by the following differential equation:

\frac{d}{dt}u_i = \frac{-u_i}{\tau} + \sum_{j=1}^n w_{ij} v_j + \theta_i

Kohonen neural network

Self Organizing Map (or Kohonen Map or SOM) is a type of Artificial Neural Network which is also inspired by biological models of neural systems from the 1970s. It follows an unsupervised learning approach and trained its network through a competitive learning algorithm. SOM is used for clustering and mapping (or dimensionality reduction) techniques to map multidimensional data onto lower-dimensional which allows people to reduce complex problems for easy interpretation. SOM has two layers, one is the Input layer and the other one is the Output layer.

The architecture of the Self Organizing Map with two clusters and n input features of any sample is given below:



How do SOM works?

Let's say an input data of size (m, n) where m is the number of training examples and n is the number of features in each example. First, it initializes the weights of size (n, C) where C is the number of clusters. Then iterating over the input data, for each training example, it updates the winning vector (weight vector with the shortest distance (e.g Euclidean distance) from training example). Weight updation rule is given by :

wij = wij(old) + alpha(t) * (xik - wij(old))

where alpha is a learning rate at time t, j denotes the winning vector, i denotes the ith feature of training example and k denotes the kth training example from the input data. After training the SOM network, trained weights are used for clustering new examples. A new example falls in the cluster of winning vectors.

Algorithm

Training:

Step 1: Initialize the weights wij random value may be assumed. Initialize the learning rate α.

Step 2: Calculate squared Euclidean distance.

$$D(j) = \Sigma (wij - xi)^2 \quad \text{where } i=1 \text{ to } n \text{ and } j=1 \text{ to } m$$

Step 3: Find index J, when D(j) is minimum that will be considered as winning index.

Step 4: For each j within a specific neighborhood of j and for all i, calculate the new weight.

$$wij(new)=wij(old) + \alpha[xi - wij(old)]$$

Step 5: Update the learning rule by using :

$$\alpha(t+1) = 0.5 * t$$

Step 6: Test the Stopping Condition

Hamming neural network

A Hamming neural network, also known as a Hamming net or Hamming network, is a type of artificial neural network designed to recognize patterns with binary inputs. It is named after Richard Hamming, who introduced the concept in the context of error-correcting codes.

Here are the key characteristics and components of a Hamming neural network:

1. Binary Input:
- Hamming networks operate on binary input patterns, where each input can take on values of 0 or 1.
2. Neuron Activation:
- Each neuron in the network corresponds to a position (bit) in the input pattern.
- The activation of a neuron is determined by the product of the input and a corresponding weight. The weights are typically set to -1 and 1.
3. Output Calculation:
- The output of the network is calculated based on the sum of the weighted inputs.
- If the sum is positive, the output is 1; if negative, the output is 0.
4. Pattern Recognition:
- Hamming networks are often used for pattern recognition tasks, especially for binary patterns.
- They are capable of recognizing patterns even in the presence of errors or noise.
5. Error Detection and Correction:
- One of the main applications of Hamming networks is in error detection and correction.
- By using redundancy in the input patterns, Hamming networks can detect and correct errors introduced during transmission or storage.
6. Hamming Distance:
- The Hamming distance between two binary patterns is the number of positions at which the corresponding bits differ.
- In the context of Hamming networks, patterns with a small Hamming distance are more likely to be recognized correctly.

7. Learning Algorithm:
- Hamming networks are typically not trained using traditional gradient-based learning algorithms like backpropagation.
- Instead, they are often implemented with fixed weights based on the desired pattern recognition or error-correction requirements.
8. Applications:
- Hamming networks have been applied in various fields, including error-correcting codes, digital communication, and pattern recognition tasks where binary patterns need to be recognized with a certain level of fault tolerance.

It's important to note that Hamming networks are relatively simple compared to more modern neural network architectures. While they may not be as versatile as deep learning models for certain tasks, they have found specific applications where the ability to handle errors and binary patterns is crucial.

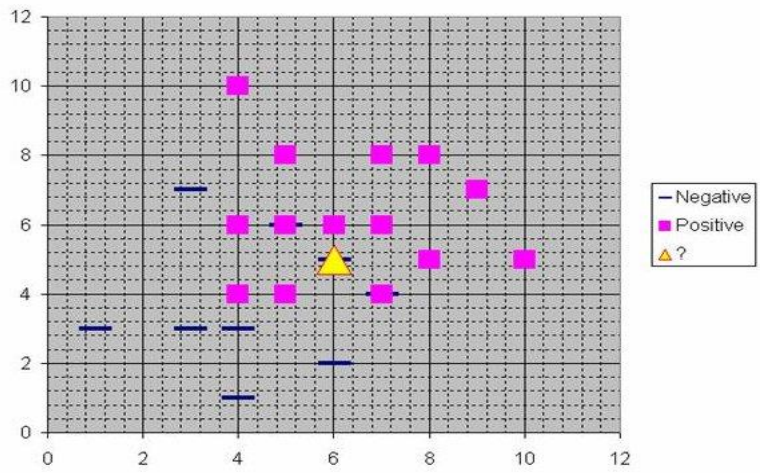
Radial basis function neural networks

A Radial Basis Function (RBF) neural network has an input layer, a hidden layer and an output layer. The neurons in the hidden layer contain Gaussian transfer functions whose outputs are inversely proportional to the distance from the center of the neuron.

RBF networks are similar to K-Means clustering and PNN/GRNN networks. The main difference is that PNN/GRNN networks have one neuron for each point in the training file, whereas RBF networks have a variable number of neurons that is usually much less than the number of training points. For problems with small to medium size training sets, PNN/GRNN networks are usually more accurate than RBF networks, but PNN/GRNN networks are impractical for large training sets.

How RBF networks work?

Although the implementation is very different, RBF neural networks are conceptually similar to K-Nearest Neighbor (k-NN) models. The basic idea is that a predicted target value of an item is likely to be about the same as other items that have close values of the predictor variables. Consider this figure:



Assume that each case in the training set has two predictor variables, x and y. The cases are plotted using their x,y coordinates as shown in the figure. Also assume that the target variable has two categories, positive which is denoted by a square and negative which is denoted by a dash. Now, suppose we are trying to predict the value of a new case represented by the triangle with predictor values x=6, y=5.1. Should we predict the target as positive or negative?

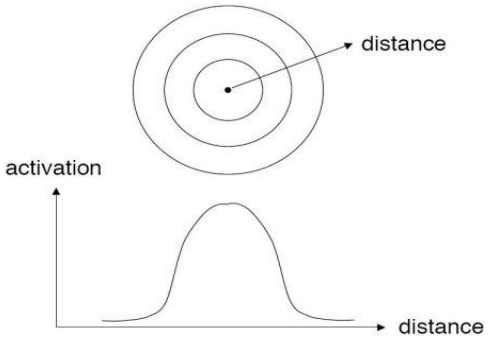
Notice that the triangle is position almost exactly on top of a dash representing a negative value. But that dash is in a fairly unusual position compared to the other dashes which are clustered below the squares and left of center. So it could be that the underlying negative value is an odd case.

The nearest neighbor classification performed for this example depends on how many neighboring points are considered. If 1-NN is used and only the closest point is considered, then clearly the new point should be classified as negative since it is on top of a known negative point. On the other hand, if 9-NN classification is used and the closest 9 points are considered, then the effect of the surrounding 8 positive points may overbalance the close negative point.

An RBF network positions one or more RBF neurons in the space described by the predictor variables (x,y in this example). This space has as many dimensions as there are predictor variables. The Euclidean distance is computed from the point being evaluated (e.g., the triangle in this figure) to the center of each neuron, and a radial basis function (RBF) (also called a kernel function) is applied to the distance to compute the weight (influence) for each neuron. The radial basis function is so named because the radius distance is the argument to the function.

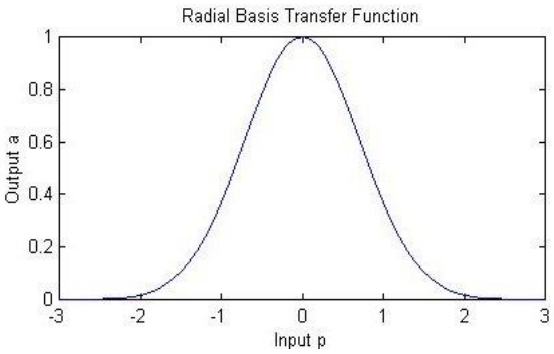
Weight = RBF(distance)

The further a neuron is from the point being evaluated, the less influence it has.

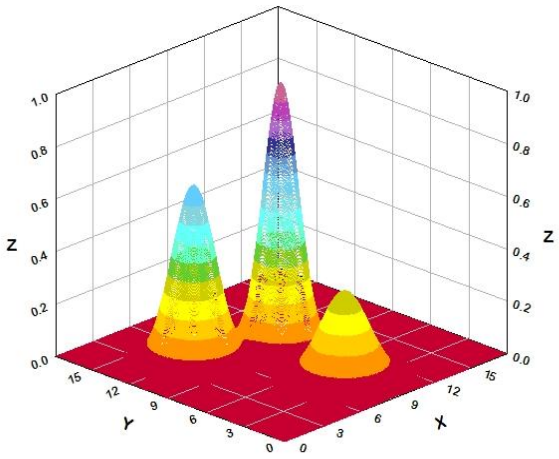


Radial Basis Function

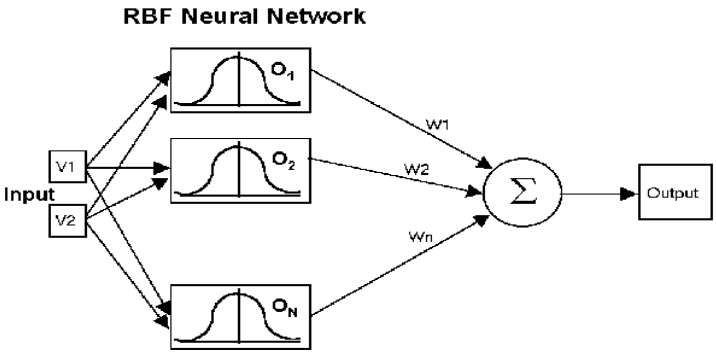
Different types of radial basis functions could be used, but the most common is the Gaussian function:



If there is more than one predictor variable, then the RBF function has as many dimensions as there are variables. The following picture illustrates three neurons in a space with two predictor variables, X and Y. Z is the value coming out of the RBF functions:



RBF Network Architecture



RBF networks have three layers:

1. Input layer – There is one neuron in the input layer for each predictor variable. In the case of categorical variables, N-1 neurons are used where N is the number of categories. The input neurons (or processing before the input layer) standardizes the range of the values by subtracting the median and dividing by the interquartile range. The input neurons then feed the values to each of the neurons in the hidden layer.

2. Hidden layer – This layer has a variable number of neurons (the optimal number is determined by the training process). Each neuron consists of a radial basis function centered on a point with as many dimensions as there are predictor variables. The spread (radius) of the RBF function may be different for each dimension. The centers and spreads are determined by the training process. When presented with the x vector of input values from the input layer, a hidden neuron computes the Euclidean distance of the test case from the neuron's center point and then applies the RBF kernel function to this distance using the spread values. The resulting value is passed to the the summation layer.
3. Summation layer – The value coming out of a neuron in the hidden layer is multiplied by a weight associated with the neuron (W1, W2, ...,Wn in this figure) and passed to the summation which adds up the weighted values and presents this sum as the output of the network. Not shown in this figure is a bias value of 1.0 that is multiplied by a weight W0 and fed into the summation layer. For classification problems, there is one output (and a separate set of weights and summation unit) for each target category. The value output for a category is the probability that the case being evaluated has that category.