Radial Basis Function

Contents

- Introduction
- Architecture
- Designing
- Learning strategies
- MLP vs RBFN

Introduction

- Radial basis functions network was formulated by Broomhead and Lowe in 1988.
- Radial basis function (RBFs) has only one hidden layer and the convergence of optimization is much faster than other neural networks, and despite having one hidden layer.
- RBF network in its simplest form is a three-layer feedforward neural network.

Introduction

- A radial basis function network is an artificial neural network that uses radial basis functions as activation functions.
- Radial basis function neural network is distinguished from other neural networks due to their universal approximation and faster learning speed.
- Radial basis functions networks have many uses, including function approximation, time series prediction, classification, and system control.

Architecture

- Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer consists of hidden neurons, and activation function of these neurons is a Gaussian function.
- Hidden layer generates a signal which corresponds to an input vector in the input layer and corresponding to this signal the network generates a response output.

Design and Development

- To generate an output, neuron process the input signal through a function called activation function of the neuron.
- In RBF activation function of hidden neuron is ϕ(x) i.e. for an input vector X output produced by the hidden neuron will be ϕ(x)

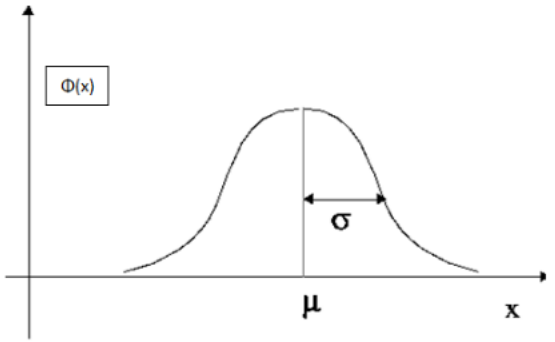


Fig-1.

$$\varphi(x) = e^{-\frac{(x-\mu)^2}{\sigma^2}}$$

Design and Development

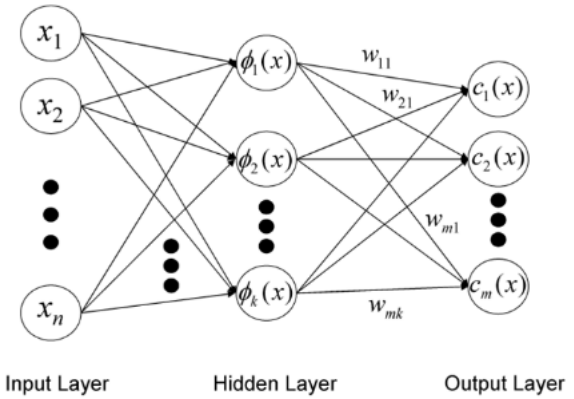
- Above figure represents a Gaussian neural activation for 1-D input x and with mean .
- Here φ(x) represents output of gaussian node given value of x.
- From above figure we can clearly see that the signal strength decreases because the input (X) moves far away from the middle.
- The basic idea of this model is that the whole feature vector space is partitioned by Gaussian neural nodes, where each node generates a sign like an input vector, and strength of the signal produced by each neuron depends on the space between its center and therefore the input vector. Also, for inputs lying closer in Euclidian vector space, the output signals that are generated must be similar.
- It works based on the principle of cover's theorem.It states that a pattern classification problem cast in a non-linear high dimensional space is more likely to be linearly separable than in a low dimensional space.

Design and Development

- Here, is center of the neuron and X is response of the neuron corresponding to input X.
- In RBF architecture, weights connecting input vector to hidden neurons represents the center of the corresponding neuron.
- These weights are predetermined such that the entire space is covered by the receptive field of these neurons, whereas values of weights connecting hidden neurons to output neurons are determined to train network.

Design and Development

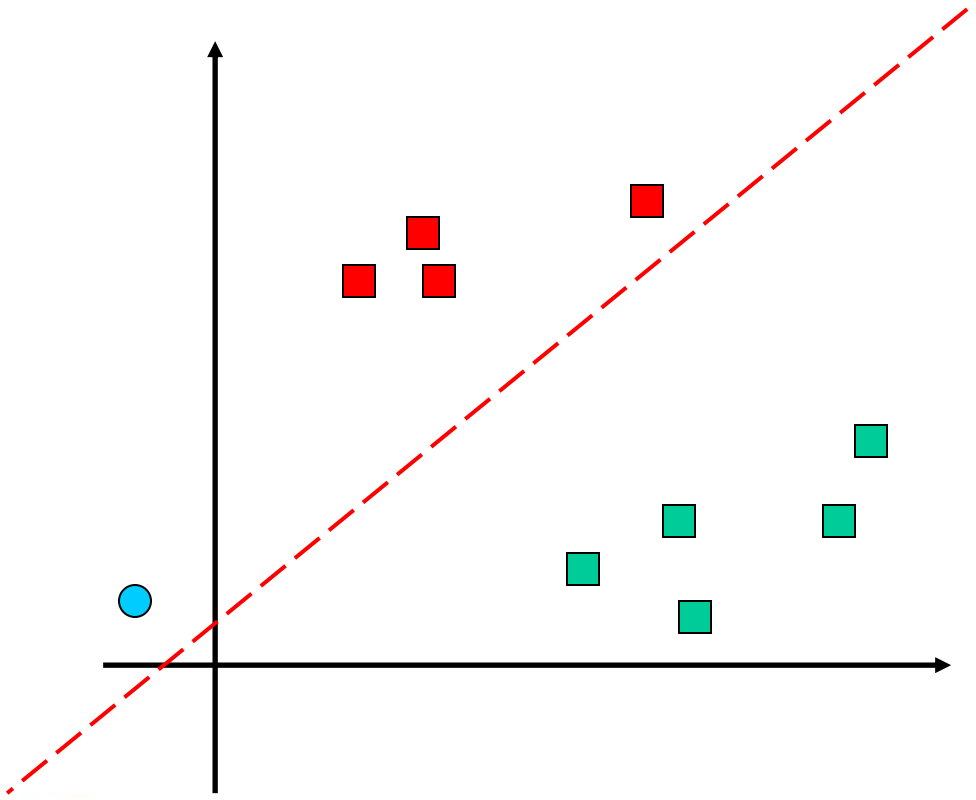
- n RBF architecture, weights connecting input vector to hidden neurons represents the center of the corresponding neuron.
- These weights are predetermined in such a way that entire space is covered by the receptive field of these neurons, whereas values of weights connecting hidden neuron to output neurons are determined to train the network.



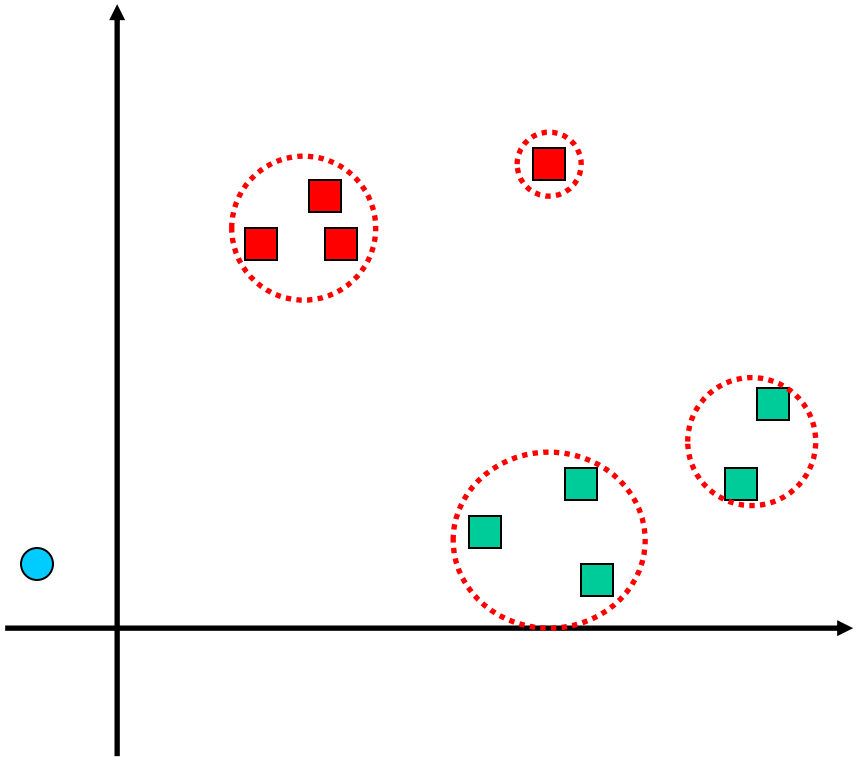
RBF vs MLP

- MLPs (Multi Layered perceptron) are better than RBFs when the underlying characteristics feature of data is embedded deep inside very high dimensional sets for example, in image recognition etc.
- RBFs have much faster convergence rate as compared to MLPs because RBFs have only one hidden layer.
- RBF network is usually preferred over MLPs for low dimensional data where deep feature extraction is not required and results are directly correlated with the components of input vector.
- Unlike most machine learning model RBF is a robust learning model.
- Classification more time in RBF than MLP.

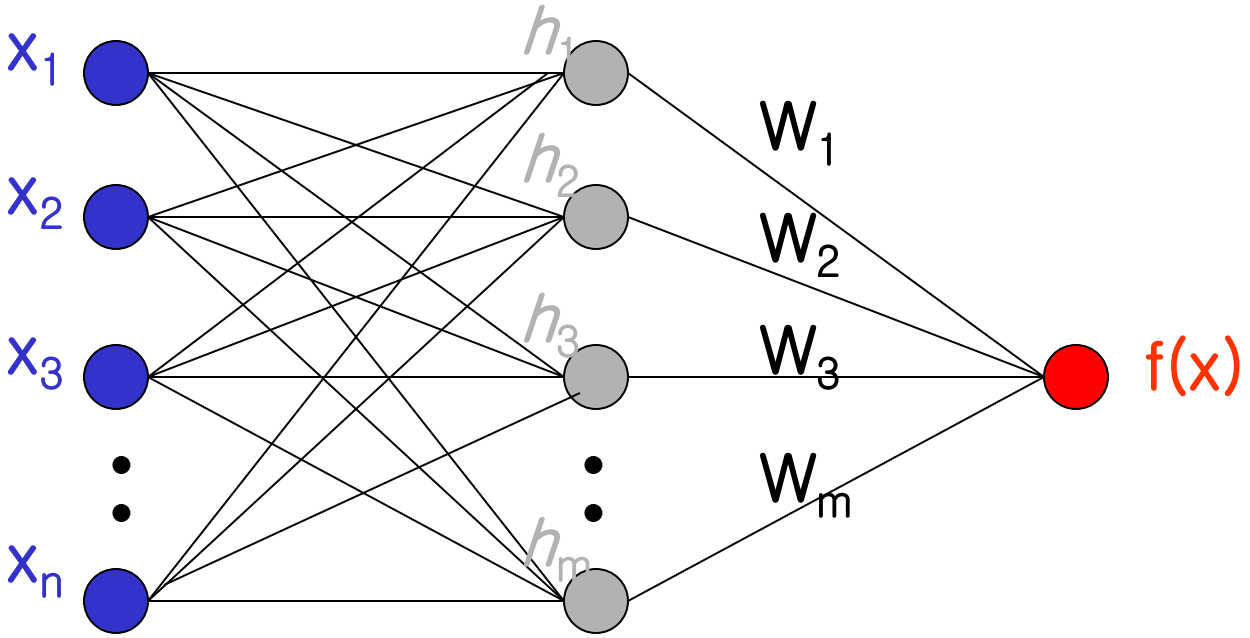
In MLP



In RBFN



Architecture



Self Organizing Map

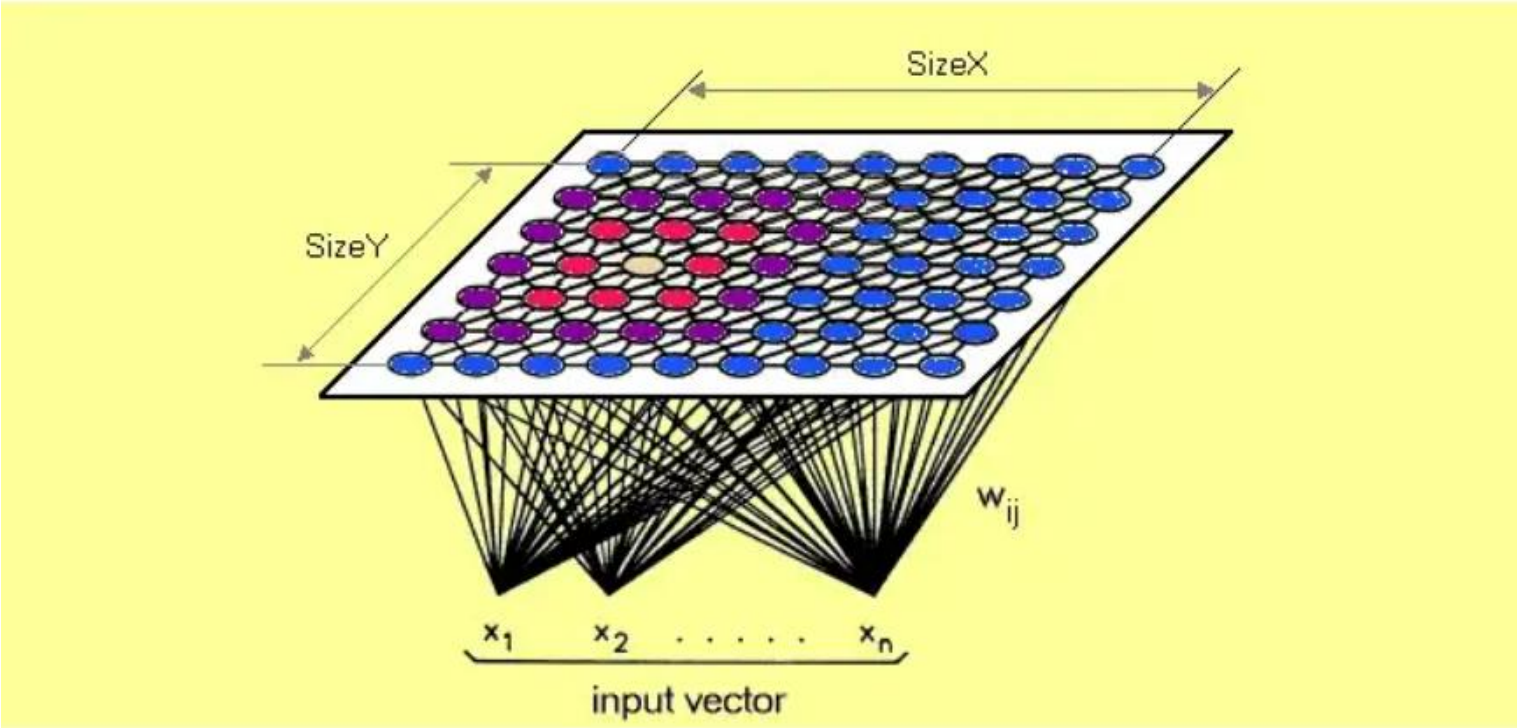
Introduction

- Self Organizing Maps or Kohenin's map is a type of artificial neural networks introduced by Teuvo Kohonen in the 1980s.
- It is an unsupervised neural network that is trained using unsupervised learning techniques to produce a low dimensional, discretized representation from the input space of the training samples, known as a map and is, therefore, a method to reduce data dimensions.

Introduction

- SOM is trained using unsupervised learning, it is a little bit different from other artificial neural networks.
- SOM doesn’t learn by backpropagation, it uses competitive learning to adjust weights in neurons.
- And it is uses in dimension reduction to reduce data by creating a spatially organized representation.
- It helps to discover the correlation between data.

SOM Architecture

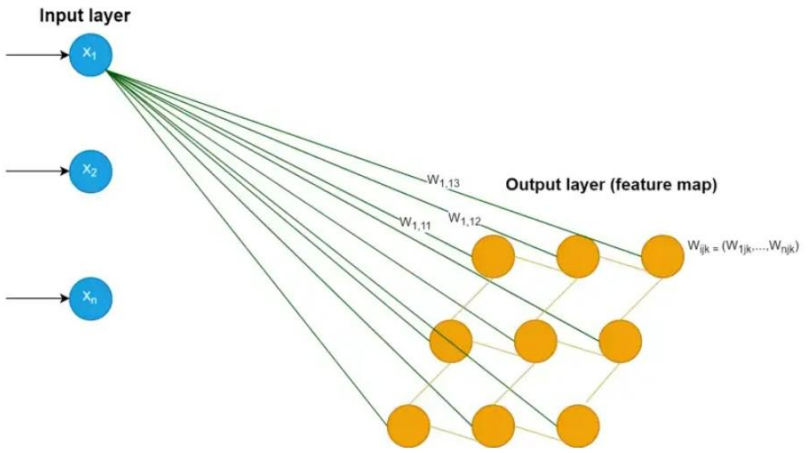


SOM Architecture

- SOM doesn’t use backpropagation with SGD to update weights, this type of unsupervised artificial neural network uses competetive learning to update its weights.
- Competetive learning is based on three processes :
  - Competetion
  - Cooperation
  - Adaptation

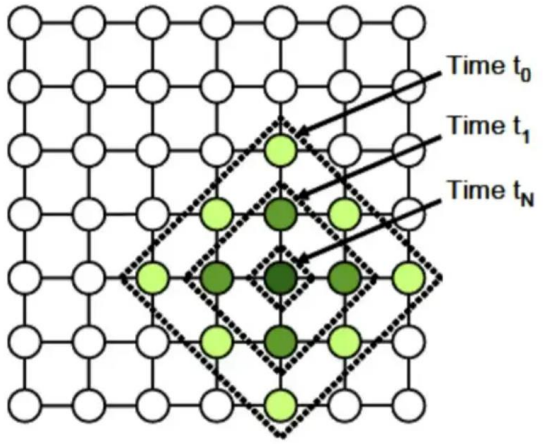
SOM Architecture : Competition

- Each neuron in a SOM is assigned a weight vector with the same dimensionality as the input space.
- In the example below, in each neuron of the output layer we will have a vector with dimension n.
- We compute distance between each neuron (neuron from the output layer) and the input data, and the neuron with the lowest distance will be the winner of the competition.
- The Euclidean metric is commonly used to measure distance.



SOM Architecture :Corporation

- The vector of the winner neuron in the final process (adaptation) is updated, but it is not the only one, also it's neighbor will be updated.
- To choose neighbors neighborhood kernel function is used.
- This function depends on two factor : time (time incremented each new input data) and distance between the winner neuron and the other neuron (How far is the neuron from the winner neuron).
- The image below show us how the winner neuron's (The most green one in the center) neighbors are choosen depending on distance and time factors.

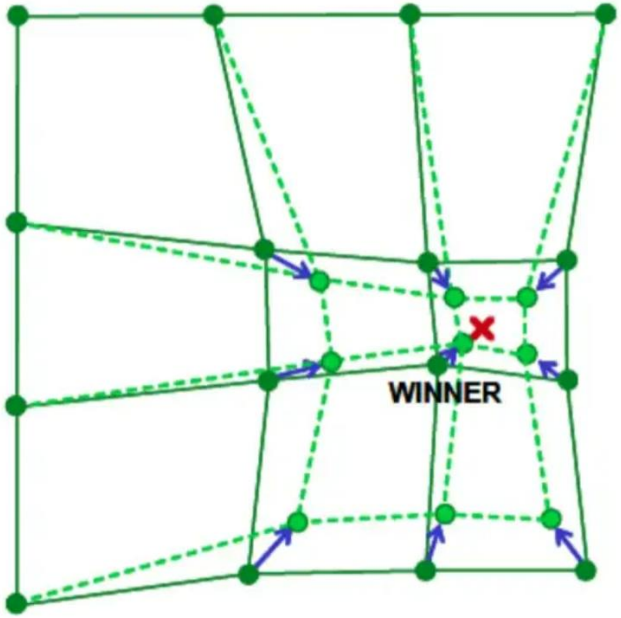


Time and distance factors

SOM Architecture :Adaptation

- After choosing the winner neuron and it's neighbors we compute neurons update.
- Those choosen neurons will be updated but not the same update, more the distance between neuron and the input data grow less we adjust it like shown in the image.
- The winner neuron and it's neighbors will be updated using this formula:
- Wij(new)=Wij(old)+ η(Xi-Wij(old))

- Where η is the learning rate (η=0 to 1)



SOM Architecture :Adaptation

- This learning rate indicates how much we want to adjust our weights.
- After time t (positive infinite), this learning rate will converge to zero so we will have no update even for the neuron winner .
- The neighborhood kernel depends on the distance between winner neuron and the other neuron

SOM Algorithm

Step 1: Initialize the weights Wij with random value, initialize the learning rate η

Step 2: Calculate squared euclidean distance

$D(j)=\Sigma(Wij-Xi)^2$ where i=1 to m, j=1 to n

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

$Wij(new)=Wij(old)+ \eta(Xi-Wij(old))$

Step 5: Test the stopping condition

Thank You!