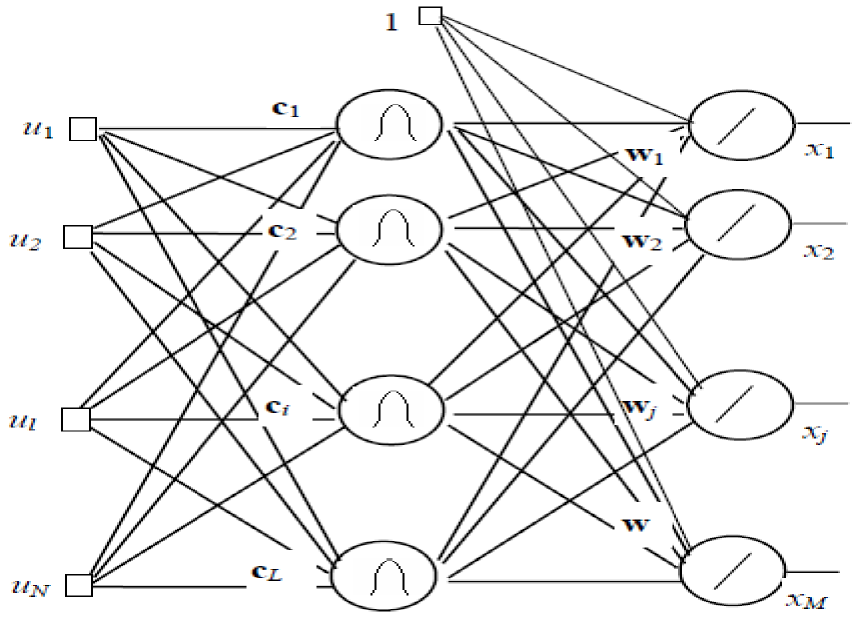
Radial Basis Function Networks

Radial basis function (RBF) networks are feed-forward networks trained using a supervised training algorithm. They are typically configured with a single hidden layer of units whose activation function is selected from a class of functions called ***basis functions.*** Each neuron in hidden layer consist of radial basis function (Gaussian)centroid on part of same dimension as the predictor variables. The output layer has a weighed sum of outputs from the hidden layer to from the network output. The major difference between RBF networks and back propagation networks (that is, multi-layer perceptron trained by Back Propagation algorithm) is the behavior of the single hidden layer.

The Structure of the RBF Networks

The structure of an RBF networks in its most basic form involves three entirely different layers (input layer , one hidden layer , and output layer):



Structure of the Standard RBF network

Input layer

The input layer is made up of source nodes (sensory units) whose number is equal to the dimension p of the input vector *u*.

Hidden layer

The second layer is the hidden layer which is composed of nonlinear units that are connected directly to all of the nodes in the input layer.

The hidden units contain a basis function, which has the parameters center and width to drive non-linearity separation . The center of the basis function for a node *i* at the hidden layer is a vector $\boldsymbol{c}_i$ whose size is the as the input vector $\boldsymbol{u}$ and there is normally a different center for each unit in the network.

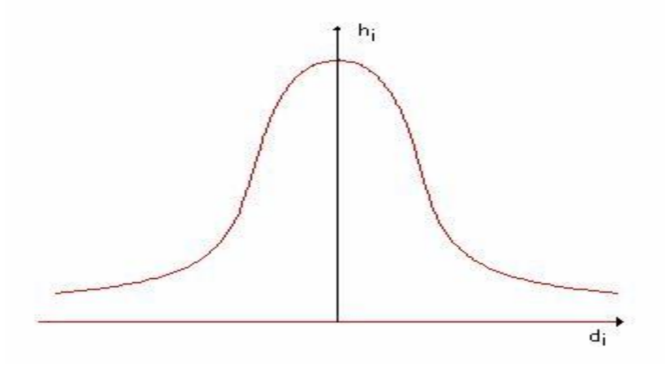
First, the radial distance d_i, meause the distance between the input vector **u** and the fixed center of the basis function $\mathbf{c}_\mathrm{i}$. the center can be one of the input . the distance metrics is usually the Euclidean distance , is computed for each unit *i* in the hidden layer as:

$$d_i = ||u - c_i||$$

The output h_i of each hidden unit *i* is then computed by applying the basis function :

$$y(x) = f(\sum_{i=}^{n} \parallel x_i - t \parallel)$$

The basis function is a curve (typically a Gaussian function, the width corresponding to the variance,σ_i) which has a peak at zero distance and it decreases as the distance from the center increases.



Base function

range of theoretical and empirical studies have indicated that many properties of the interpolating function are relatively insensitive to the precise form of the basis functions .Some of the most commonly used basis functions are: , for the Gaussian Functions:

$$\emptyset(r) = \exp(r\hat{}2/(2\sigma\hat{}2\,)) \qquad \text{with parameter } \sigma > 0$$

$$\emptyset(r) = \exp((||u - c\_i\,||)/(2\sigma\hat{}2\,))$$

Any clustering algorithm can be used to determine the RBF unit centers (e,g, K-means clustering)
K-Mean Algorithm:

1. Choose a set of centers $c_1, \ldots\ldots c_k$ arbitrarily
2. Assign N samples to the K subsets using the min Euclidean distance rule:

$$x^p \in c_i \; if \; ||x^p - c_i|| < || \, \Big| |x^p - c_j| \Big| \quad \text{fo r all point}$$

3. After all data points are assigned go to next step
4. Compute new subset center (ci) s.t. min the cost function

$$J_i = \sqrt{\sum \|x_i - c_i\|^2}$$

5If any subset center is changed, return to step 2, otherwise stop.

Output layer

The transformation to the hidden unit space to the output space is linear. The jth output is computed as

$$\boldsymbol{x_j = f_j(u) = \sum_{i=1}^{L} w_{ij} h_i \qquad j = 1, 2, \ldots\ldots M}$$

Mathematical model

In summary, the mathematical model of the RBF network can be expressed as:

$$\boldsymbol{x_j = f_j(u) = \sum_{i=1}^{L} w_{ij} G(\|u - c_i\|) \qquad j = 1, 2, \ldots\ldots M}$$

Where is the Euclidean distance between **u** and **c***i*

Function approximation

Let y=g(u) be a given function of u, y∈R, u∈R, g:R→R, and let G*i* *i*=1..L, be a finite set of basis functions. The function g can be written in terms of the given basis functions as

$$\boldsymbol{y = g(u) = \sum_{i=1}^{L} w_i g_i(u) + r(u)}$$

where r(*u*) is the residual.

The function y can be approximated as

$$\boldsymbol{y = g(u) = \sum_{i=1}^{L} w_i G_i(u)}$$

The aim is to minimize the error by setting the parameters of G*i* appropriately. A possible choice for the error definition is the L2 norm of the residual function r(u) which is defined as

$$\boldsymbol{||r(u)||_{L_2} = \int r(u)^2}$$

Approximation by RBFNN

Now, consider the single input single output RBF network shown in Figure 4. Then *y* can be written as:

$$\boldsymbol{y = g(u) = \sum_{i=1}^{L} w_i g_i(u) + r(u)}$$

By the use of such a network, y can be written as

$$\boldsymbol{y = \sum_{i=1}^{L} w_{ij} G(\|u - c_i\|) + r(u) = f(u) + r(u)}$$

Where f(*u*) is the output of the RBFNN given in Figure and r(*u*) is the residual. By setting the center c*i,* the variance σ$_i$, and the weight *wi* the error appropriately, the error can be minimized.

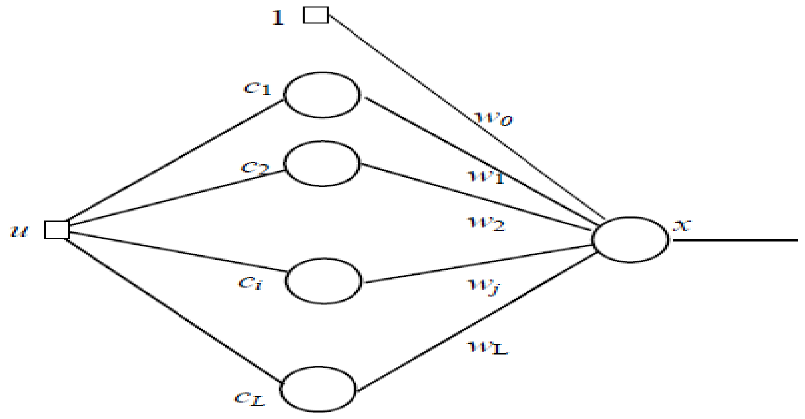


Figure 4: Single input, single output RBF network

Whatever we discussed here for g:R→R, can be generalized to **g**:R^N→R^M easily by using an *N* input, **M** output RBFNN given in figure 9.1 previously.

Data Interpolation

Given input output training patterns ($\mathbf{u}^k$,$\mathbf{y}^k$), *k*=1,2, ..*K*, the aim of data interpolation is to approximate the function **y** from which the data is generated. Since the function **y** is unknown, the problem can be stated as a minimization problem which takes only the sample points into consideration:
Choose *wi,j* and **c***i*, *i*=1,2...*L*, *j*=1,2...*M* so as to minimize:

$$\boldsymbol{J}(\boldsymbol{w},\boldsymbol{c}) = \sum_{\boldsymbol{k}=1}^{\boldsymbol{K}} ||\boldsymbol{Y}^{\boldsymbol{k}} - \boldsymbol{f}(\boldsymbol{u}^{\boldsymbol{k}})||^{\boldsymbol{2}}$$

As an example, the output of an RBF network trained to fit the data points given in Table is given in Figure 95.

TABLE I: 13 data points generated by using sum of three gaussians with c_1=0.2000 c_2=0.6000 c_3=0.9000 w_1=0.2000 w_2=0.5000 w_3=0.3000 σ=0.1000

data no	1	2	3	4	5	6	7	9	10	11	12	13
x	0.0500	0.2000	0.2500	0.3000	0.4000	0.4300	0.4800	0.6000	0.7000	0.8000	0.9000	0.9500
f(*x*)	0.0863	0.2662	0.2362	0.1687	0.1260	0.1756	0.3290	0.6694	0.4573	0.3320	0.4063	0.3535

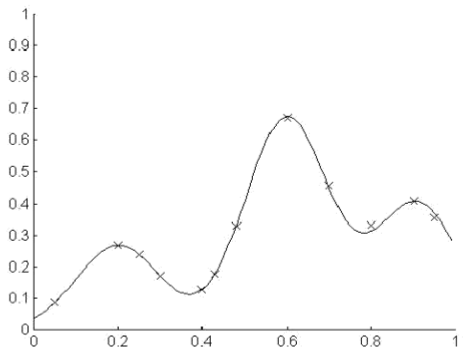


Figure 5 Output of the RBF network trained to fit the datapoints given in Table

Training RBF Networks

The training of a RBF network can be formulated as a nonlinear unconstrained optimization problem given below:

Given input output training patterns ($\mathbf{u}^k$,$\mathbf{y}^k$), k=1,2, ..K, choose $w_{i,j}$ and $\mathbf{c}_i$,, i=1,2...L, j=1,2...M so as to minimize

$$\boldsymbol{J(w,c)} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{K}} ||\boldsymbol{Y^k} - \boldsymbol{f(u^k)}||^{\mathbf{2}}$$

Note that the training problem becomes quadratic once if **c**i's (radial basis function centers) are known.

Adjusting the widths

In its simplest form, all hidden units in the RBF network have the same width or degree of sensitivity to inputs. However, in portions of the input space where there are few patterns, it is sometime desirable to have hidden units with a wide area of reception. Likewise, in portions of the input space, which are crowded, it might be desirable to have very highly tuned processors with narrow reception fields. Computing these individual widths increases the performance of the RBF network at the expense of a more complicated training process.

Adjusting the centers

Remember that in a back propagation network, all weights in all of the layers are adjusted at the same time. In radial basis function networks, however, the weights into the hidden layer basis units are usually set before the second layer of weights is adjusted. As the input moves away from the connection weights, the activation value falls off. This behavior leads to the use of the term "center" for the first-layer weights. These center weights can be computed using Kohonen feature maps, statistical methods such as K-Means clustering, or some other means. In any case, they are then used to set the areas of sensitivity for the RBF network's hidden units, which then remain fixed.

3 Adjusting the weights

Once the hidden layer weights are set, a second phase of training is used to adjust the output weights. This process typically uses the standard steepest descent algorithm. Note that the training problem becomes quadratic once if **c**i's (radial basis function centers) are known.

Self-Organizing Maps

Topology Preserving Maps

- A topology is a system of subsets $O \subset X$ called neighborhoods.
- In a metric space (X, d), the neighborhood $O_\varepsilon(x)$ of point x can be defined as a set of points a such that

  $d(x, a) < \varepsilon$
- If X and Y are two topological spaces, then functions $f : X \to Y$ that preserve the topology of X in Y are continuous functions.
- Continuous functions map some neighborhood $O(x) \subseteq X$ into any neighborhood $O(y) \subseteq Y$:

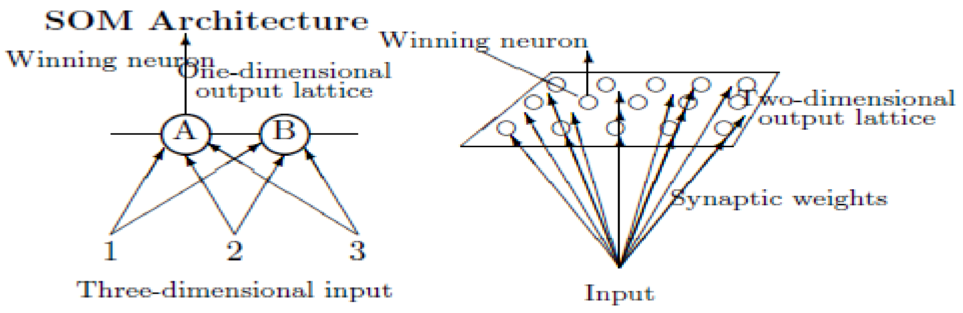
  $f(x) = y$ and $f(O(x)) \subseteq O(y)$
- Thus, to visualize data we need a continuous function from m-dimensional space into a 2-dimensional space:

  $f : R^m \to R^2$

- Remark 1. If the data 'lives' in a m-dimensional space with m > 3, then there A topology is a system of subsets O ⊂ X, called neighborhoods.

Self-Organising Maps

1 SOM Architecture



- A Self-Organizing Map (SOM) is an unsupervised neural network algorithm that learns a topology preserving map, and it is used to visualize high-dimensional data.
- It uses a single layer network of artificial neurons (or nodes), which are arranged into k × l grid and are connected to the input space (data).
- The SOM algorithm is designed to establish a correspondence between topologies of the input space (data) and the output lattice (aka Kohonen's topology preserving map).
- Each node has m weights, and together the weights represent some m-dimensional vector:

$$\mathbf{w_j = (w_{1j}, \ldots, w_{mj}) \in R^m} .$$

- The weight vector **w** of each neuron can be compared to an m-dimensional vector

$\mathbf{x = (x_1, \ldots, x_m)} \in R^m$ from the input space (data)

- This comparison is usually done using the Euclidean metric in the input space R^m

$$d_{in}(x, w_j) = \sqrt{|x_1 - w_{1j}|^2 + \cdots + |x_m - w_{mj}|^2}$$

- The nodes are arranged into a k × l grid (output space).
- The topology in the grid is defined by another metric, such as the taxi-cab distance:

$$d_{out}(i, j) = |i_1 - j_1| + |i_2 - j_2|$$

- The algorithm involves three phases: competition, adaptation and cooperation.

SOM Algorithm

1. Define the size of the output lattice (i.e. k × l).
2. Initialize the weights of the nodes (e.g. randomly) be assumed ,initialize the learning rate(α).
3. Define metrics d_{in}, d_{out} and other parameters (*α, h)*
4. Repeat

   - Select an input vector x € R_m from data

- Find the winning node by minimizing ***d*$_{in}$*(x,wj)***

- Adapt the weights of the winner and its neighbors

5. until the network stabilizes

Competition

- An input vector **x = (x$_1$, . . . , x$_m$)** is compared with the weight vector **w$_j$ = (w$_{1j}$, . . . , w$_{mj}$)** of each node by computing the distance d(x, w$_j$):

$$d(x, w_1) = \sqrt{(x_1 - w_{11})^2 + \cdots + (x_m - w_{m1})^2}$$

$$\vdots$$

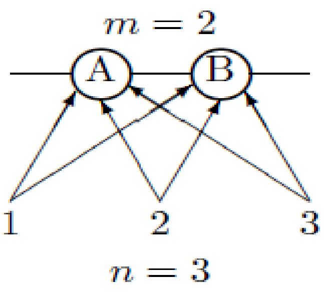
$$d(x, w_n) = \sqrt{(x_1 - w_{1n})^2 + \cdots + (x_m - w_{mn})^2}$$

- The winner is the node with the weight w$_j$ closest to the input x (i.e. shortest d(x, w$_j$)).
- Thus, nodes 'compete' in the sense which of the nodes w$_j$ is more 'similar' to a given input pattern x.

Example 1. Consider SOM with three inputs and two output nodes (A and B).
Let

w$_A$ = (2, −1, 3) , **w$_B$ = (−2, 0, 1)**

Find which node is the winner for the input
x = (1, −2, 2)



$$d(\mathbf{x}, \mathbf{w}_A) = \sqrt{(1-2)^2 + (-2+1)^2 + (2-3)^2} = \sqrt{3}$$
$$d(\mathbf{x}, \mathbf{w}_B) = \sqrt{(1+2)^2 + (-2-0)^2 + (2-1)^2} = \sqrt{14}$$

• Node A is the winner because it is 'closer' ($\sqrt{3} < \sqrt{14}$)
What if x = (−1, −2, 0)>

Adaptation

After the input x has been presented to SOM, the weights of all nodes are adapted, so that they become more 'similar' to the input x vector.

- The adaptation formula for node j is:

$$w_j^{new} = w_j^{old} + \alpha h_{ij}\left[x - w_j^{old}\right]$$

where

– $\boldsymbol{w_j}$ is the weight vector of node j ∈ [1, . . . , k × l];

– $\boldsymbol{\alpha}$ is the learning rate coefficient;

– $\boldsymbol{h_{ij}}$ is the neighborhood of node j with respect to the winner i.

Adaptation (cont.)

To understand better the adaptation formula, let us check how the weights change for different values of α and h_{ij}.

$$\boldsymbol{w_j^{new} = w_j^{old} + \alpha h_{ij}\left[x - w_j^{old}\right]}$$

- Suppose $\boldsymbol{\alpha}$ =0 or $\boldsymbol{h_{ij}}$= 0

$$\boldsymbol{w_j^{new} = w_j^{old} + 0.0\left[x - w_j^{old}\right] \;=\; w_j^{old}}$$

The weight does not change $\boldsymbol{(w_j^{new} = w_j^{old})}$

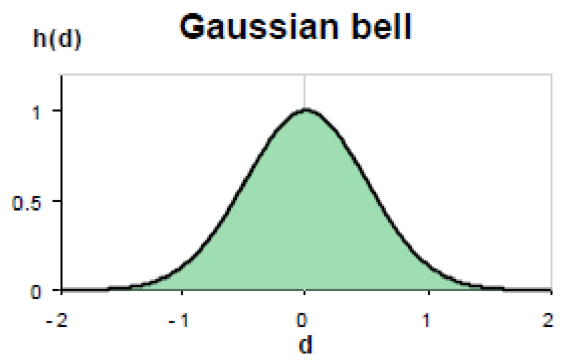
- Suppose $\boldsymbol{\alpha}$ =1 or $\boldsymbol{h_{ij}}$= 1

$$\boldsymbol{w_j^{new} = w_j^{old} + \left[x - w_j^{old}\right] \;=\; x}$$

The new weight is equal to the input ($\boldsymbol{w_j^{new} = x}$)

Cooperation

- Although weights of all nodes are adapted, they do not adapt equally. Adaptation depends on how close the nodes are from the winner in the output lattice.
- If the winner is node i, then the level of adaptation for node j is defined by the neighborhood function $h_{ij} = h(d(i, j))$, where d(i, j) is the distance in the lattice.
- The neighborhood is defines in such a way that it is smaller as the distance d(i, j) gets larger. For example, the Gaussian bell function



• The winner 'helps' mostly its neighbours to adapt. Note also that the winner is adapted more than any other node (i.e. because d(i, i) = 0).

Example 2. Let α = 0.5 and h = 1, and let us adapt the winning node A from previous example:

$$\mathbf{w}_A = (2, -1, 3)\,, \qquad \mathbf{x} = (1, -2\ ,2)$$

the adaptation formula:

$$w_j^{new} = w_j^{old} + \alpha h_{ij}\left[x - w_j^{old}\right]$$

$$\begin{aligned} \mathbf{w}_A &= \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix} + 0.5 \cdot 1 \cdot \left[\begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix} - \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix} \right] \\ &= \begin{pmatrix} 2 \\ -1 \\ 3 \end{pmatrix} + 0.5 \cdot \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ -1.5 \\ 2.5 \end{pmatrix} \end{aligned}$$