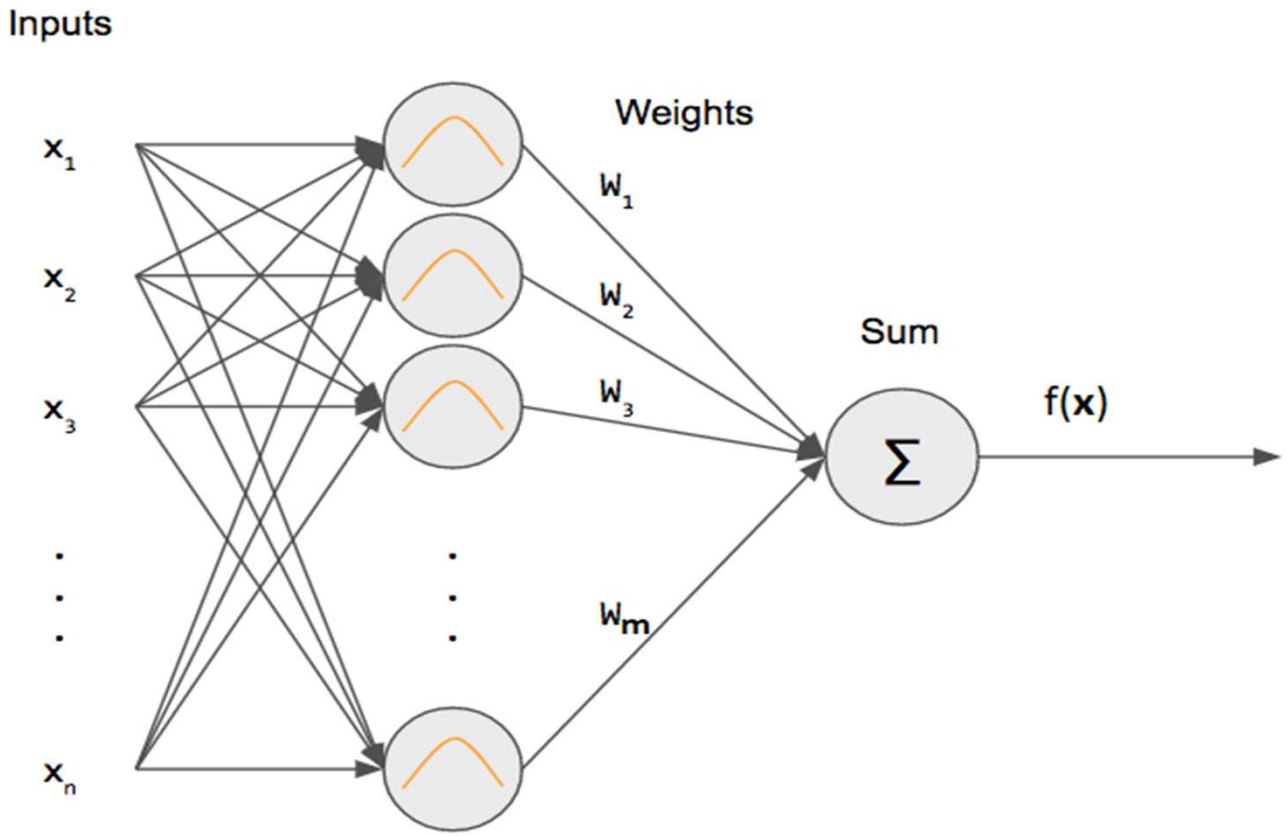
5. Radial Basis Function (RBF) Neural Networks

In the nervous system of biological organisms, there is evidence that responses of some neurons are "local" or tuned to some region of input space. An example is the orientation sensitive cells of the visual cortex, whose response is sensitive to local region in the retina. In this part, we will explore a type of neural network inspired by this finding.

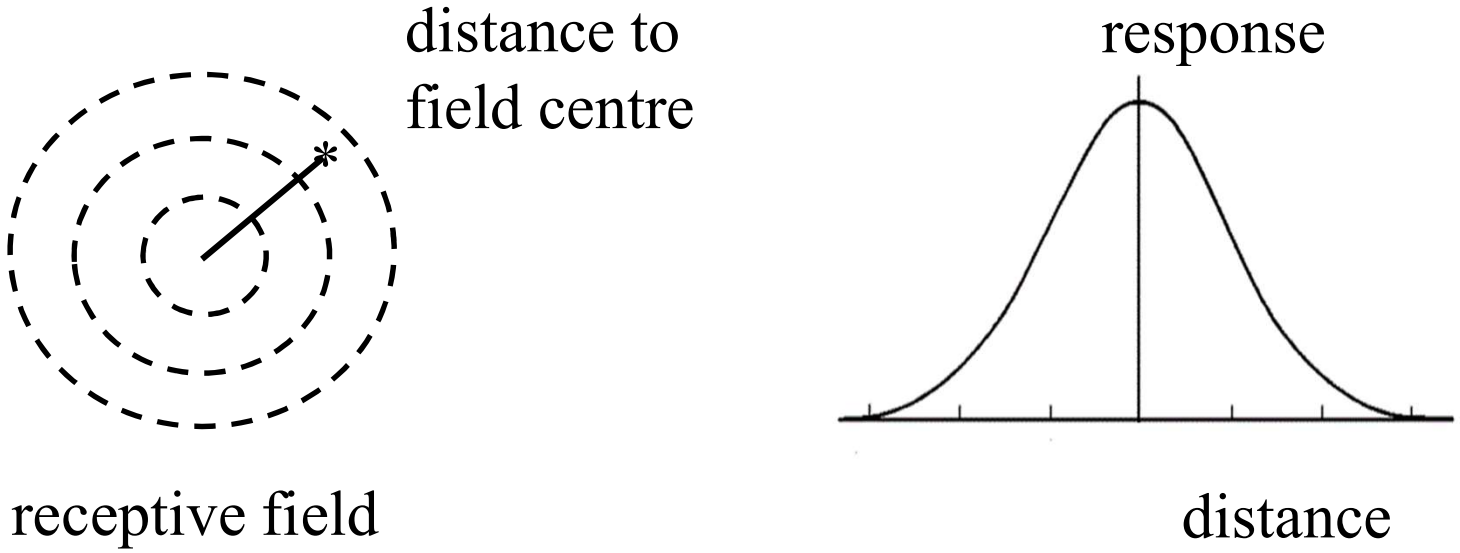
This neural network is known as the radial basis function (RBF) neural network. The RBF neural network has a feed-forward structure with a modified hidden layer and training algorithms.

RBF Neural Network Structure



The architecture of RBF neural network with a single output

As mentioned, RBF networks emulate the behavior of certain biological networks. Basically, the hidden layer consists of the locally tuned or locally sensitive neurons, whose response (output) is localized and decays as a function of the distance from the input to the center of neuron's receptive field as illustrated below:



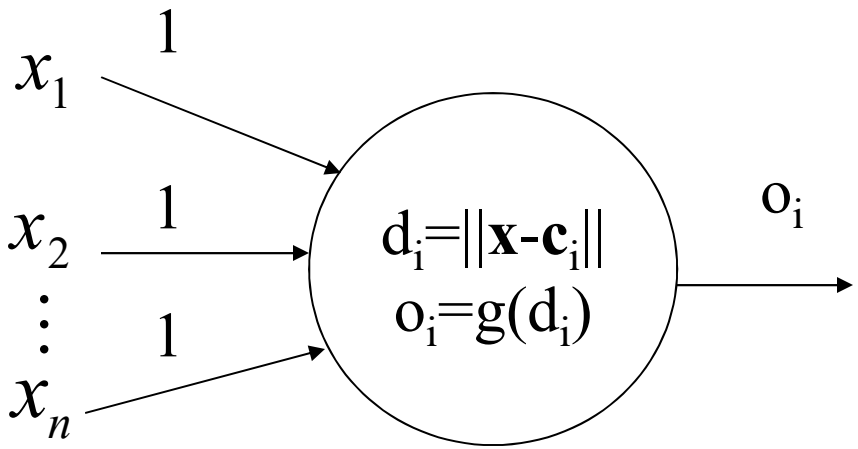
RBF Neuron Characteristics

Input layer:

The input layer neurons do not perform any computation and just distribute the input variables to the hidden layer. But note the weights between input layer neurons and hidden layer neurons in the RBF network are all set to 1.

Hidden layer:

A general form for hidden layer neurons of the RBF neural network is illustrated as:



In mathematic terms, we can describe the operation of the hidden layer neuron as:

$$d_i = \|\mathbf{x} - \mathbf{c}_i\|$$

$$o_i(\mathbf{x}) = g(d_i) = g(\|\mathbf{x} - \mathbf{c}_i\|)$$

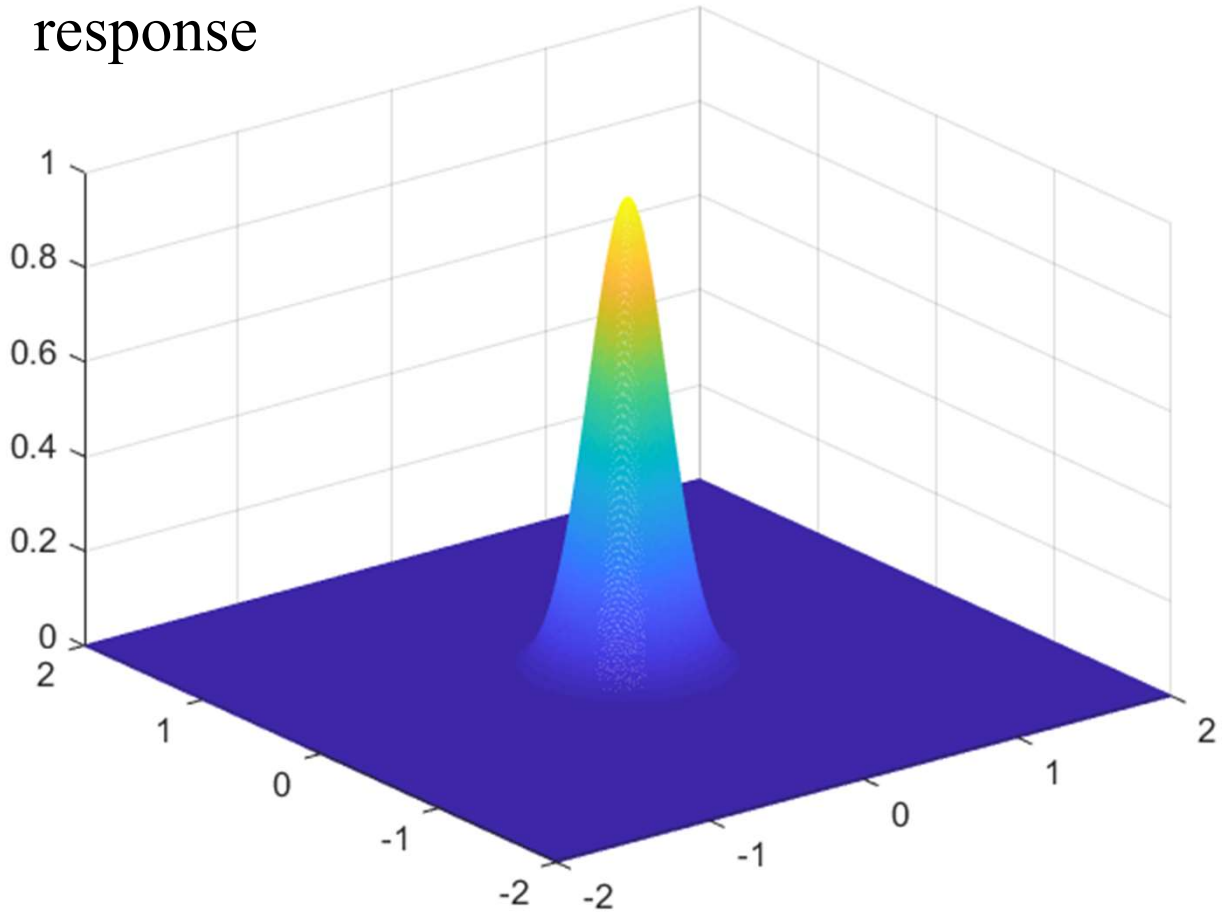
Where $\mathbf{c}_i$ is called center vector, i.e. the center of receptive field, of neuron i; g is the radial basis function. Some commonly used basis functions are as follows:

(1) The Gaussian function

$$g(d_i) = \exp\left(-\frac{d_i^2}{2\sigma^2}\right)$$

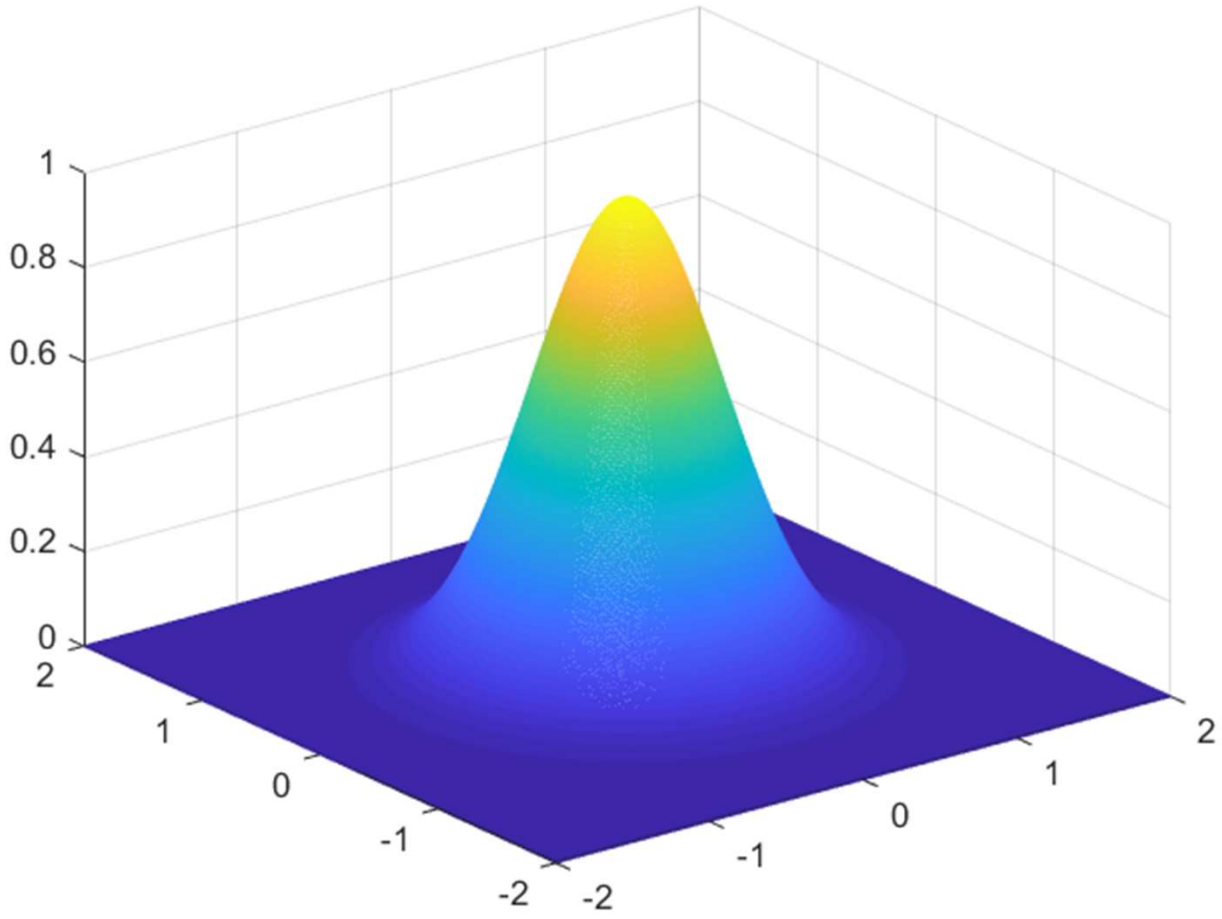
where σ is the width of the basis function, which directly determines the size of the receptive field.

Gaussian basis function with $\sigma=0.2$

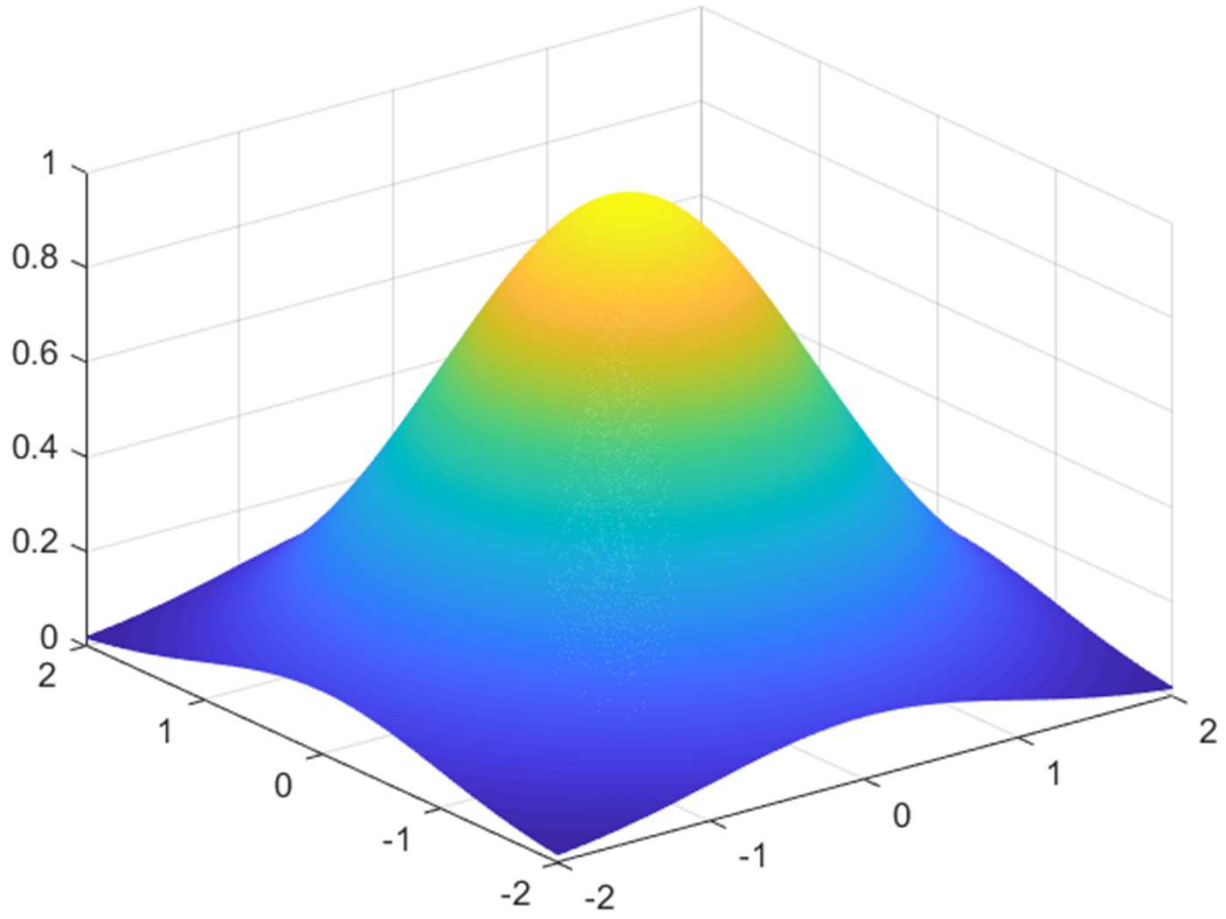


Receptive field centred at (0,0)

Gaussian basis function
with $\sigma=0.5$



Gaussian basis function with $\sigma=1$



(2) The inverse multi-quadratic function

$$g(d_i) = 1 / \sqrt{d_i^{\,2} + \sigma^2}$$

The output layer:

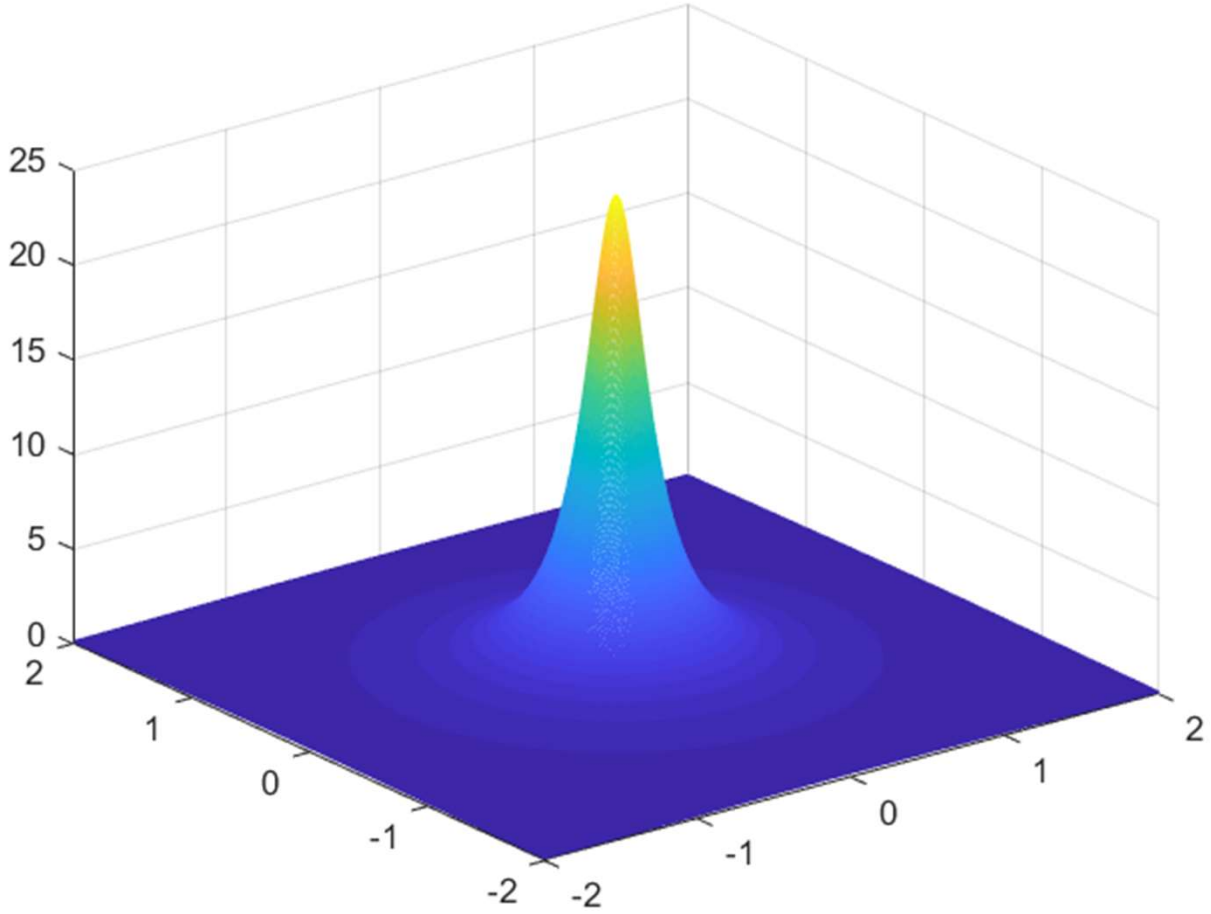
The output layer neuron is a linear combiner. The output of the network is the weighted sum of the hidden layer neuron outputs:

$$\mathrm{f}(\mathbf{x}) = \sum_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{w}_{\mathrm{j}} \mathrm{o}_{\mathrm{j}}(\mathbf{x})$$

Where m is the number of hidden layer neurons, w_j is the weight from hidden layer neuron j to the output layer neuron.

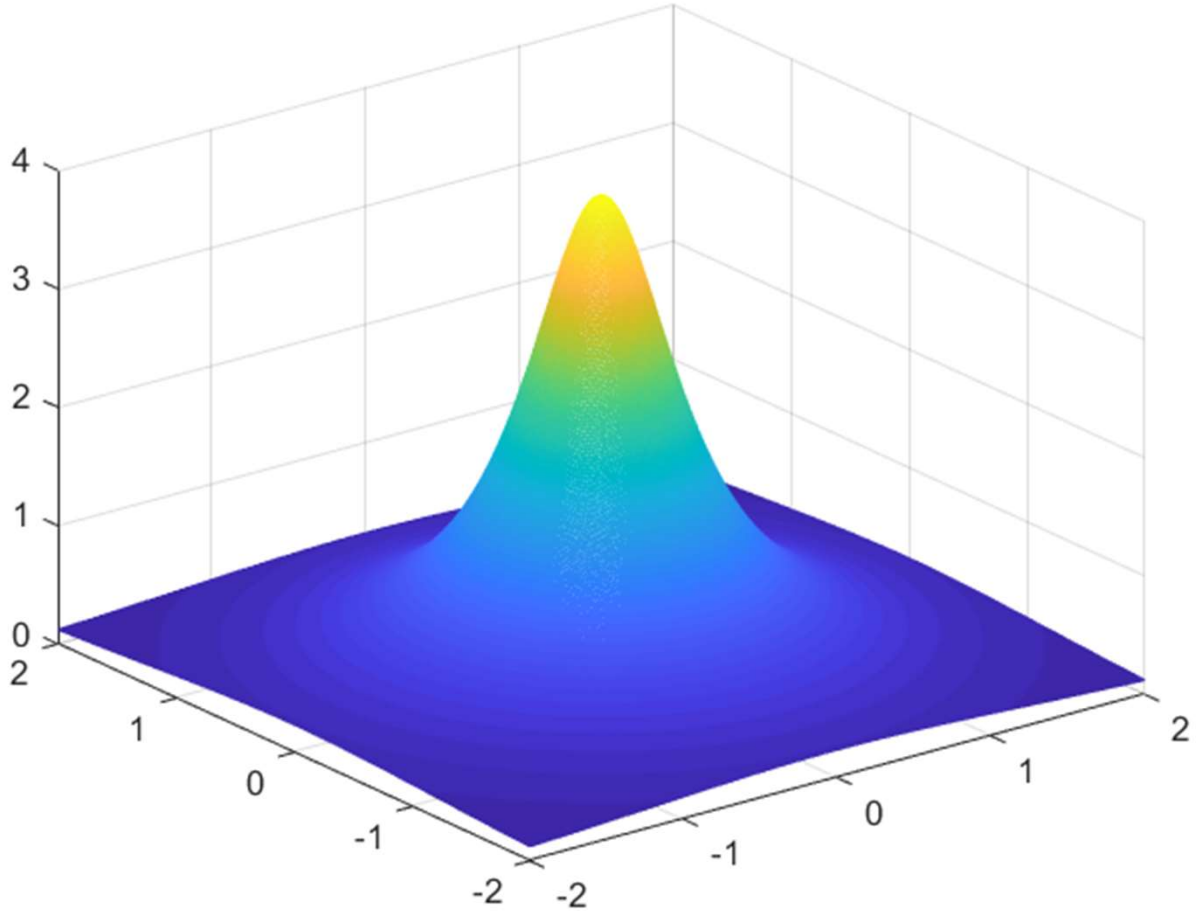
inverse multi-quadratic

σ=0.2



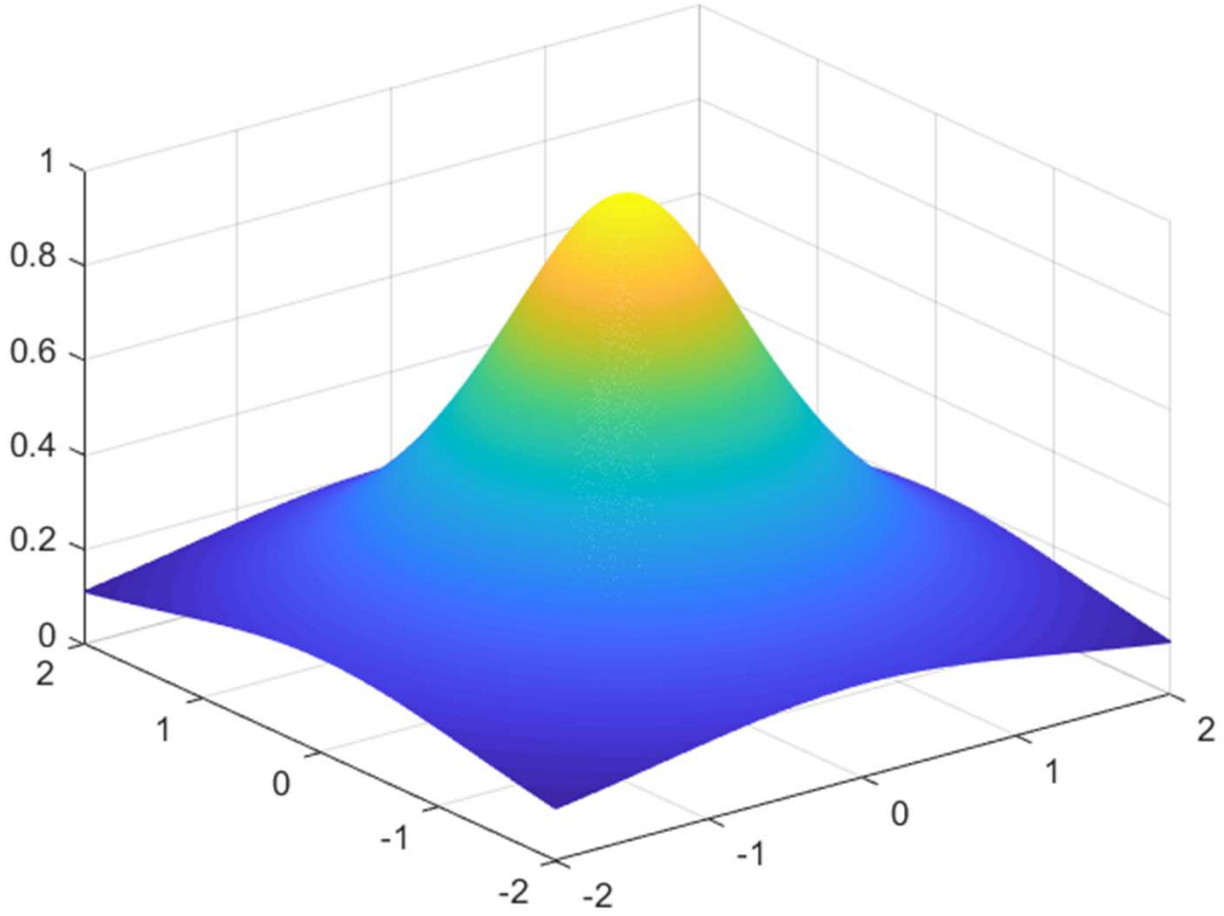
inverse multi-quadratic

σ=0.5



inverse multi-quadratic

$\sigma=1$



If Gaussian or inverse multi-quadratic function is used, we have:

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j o_j(\mathbf{x}) = \sum_{j=1}^{m} w_j \exp\left(-\frac{\left\| \mathbf{x} - \mathbf{c}_j \right\|^2}{2\sigma^2} \right)$$

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j o_j(\mathbf{x}) = \sum_{j=1}^{m} w_j \left(\left\| \mathbf{x} - \mathbf{c}_j \right\|^2 + \sigma^2 \right)^{-\frac{1}{2}}$$

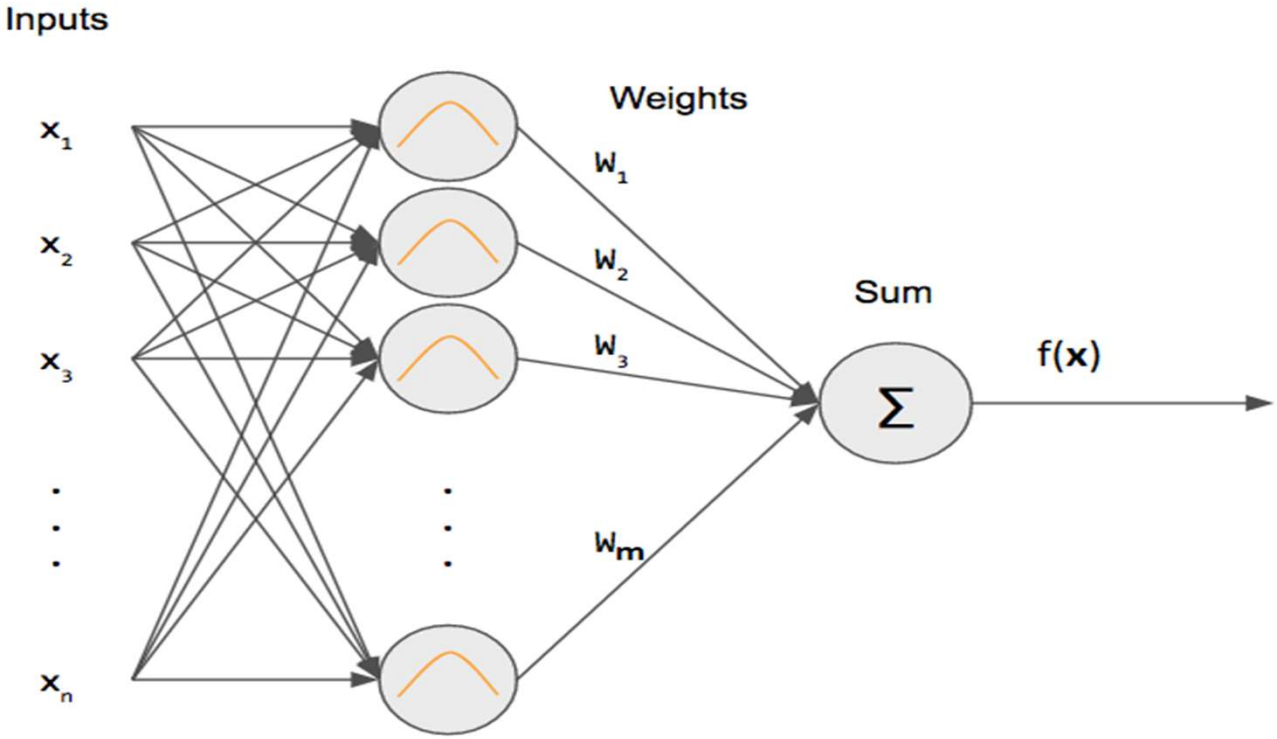
In either case, the parameters that determine the RBF are:

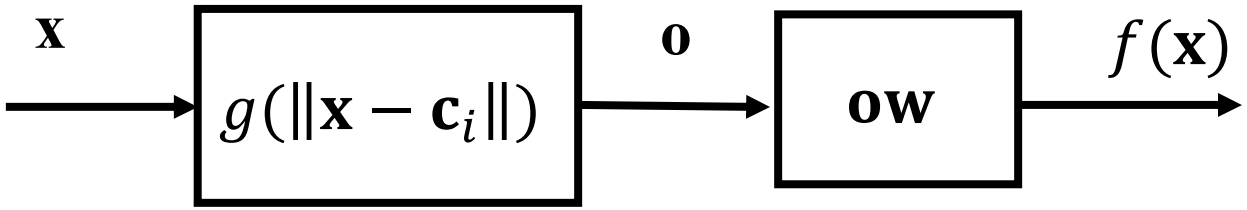
(1) The center vectors $\mathbf{c}_j$, j=1,2,…,m.

(2) The weights w_j, j=1,2,…,m.

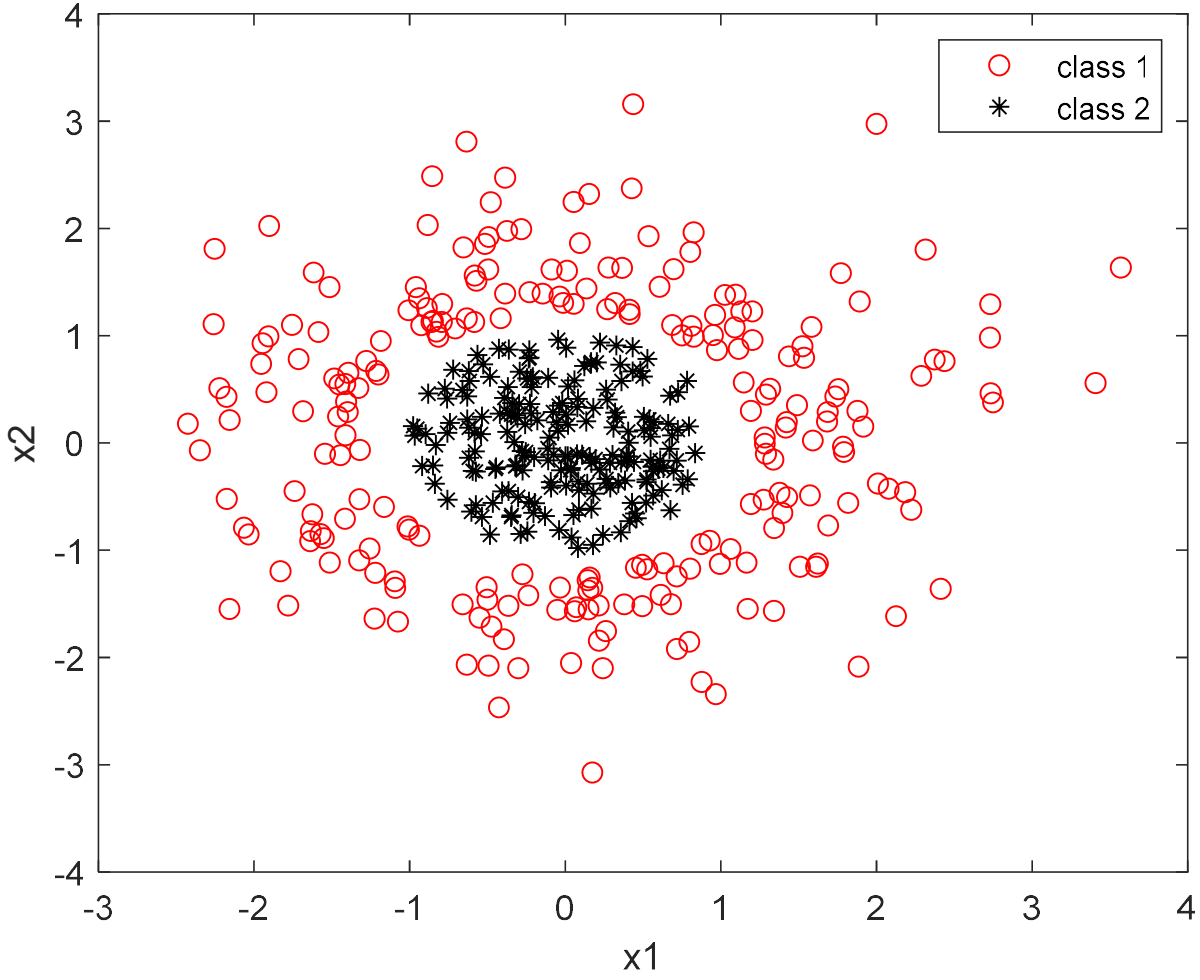
(3) The width of the basis function σ.

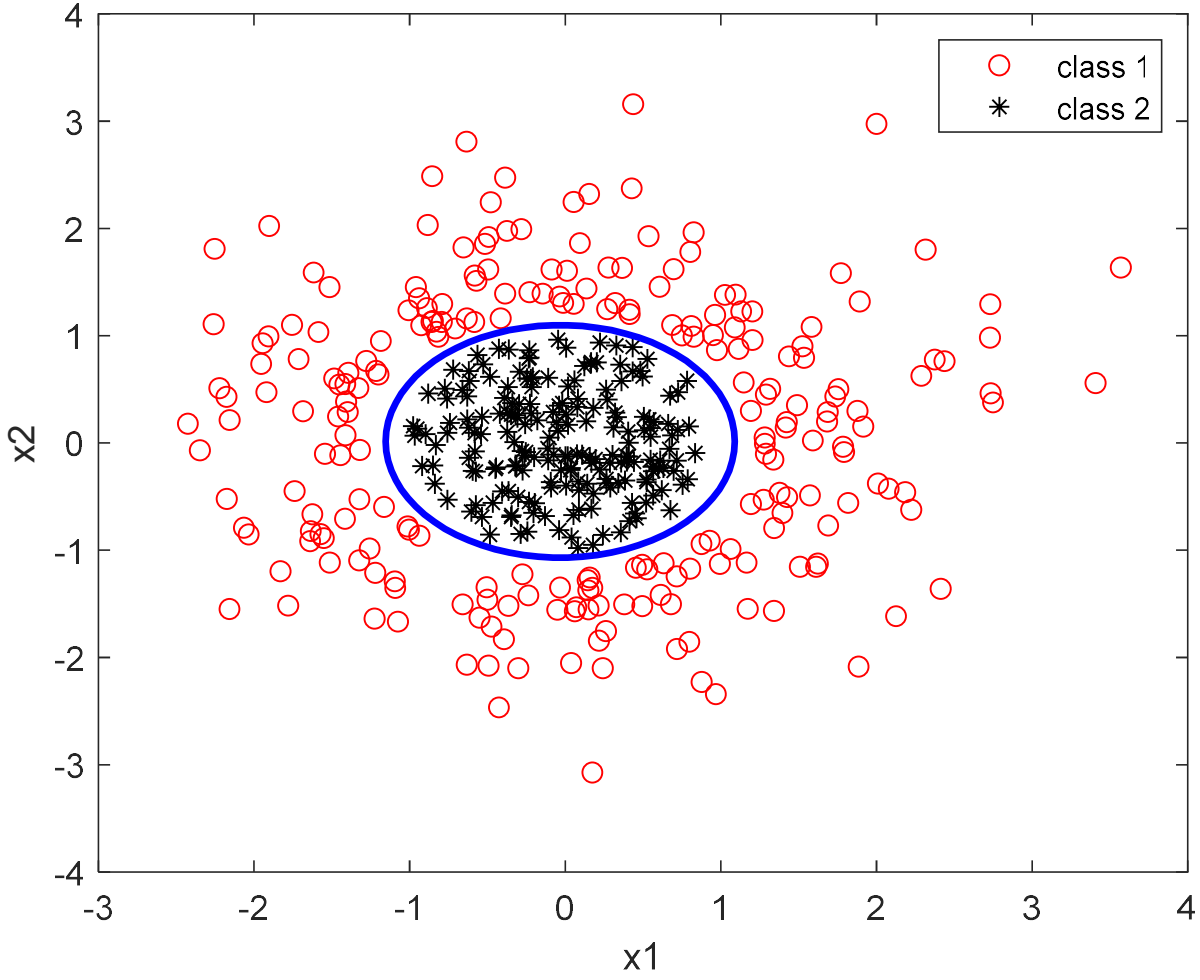
It is noted the network output is a nonlinear function of σ and $\mathbf{c}_j$ but it has a linear relation with the output of hidden neurons.

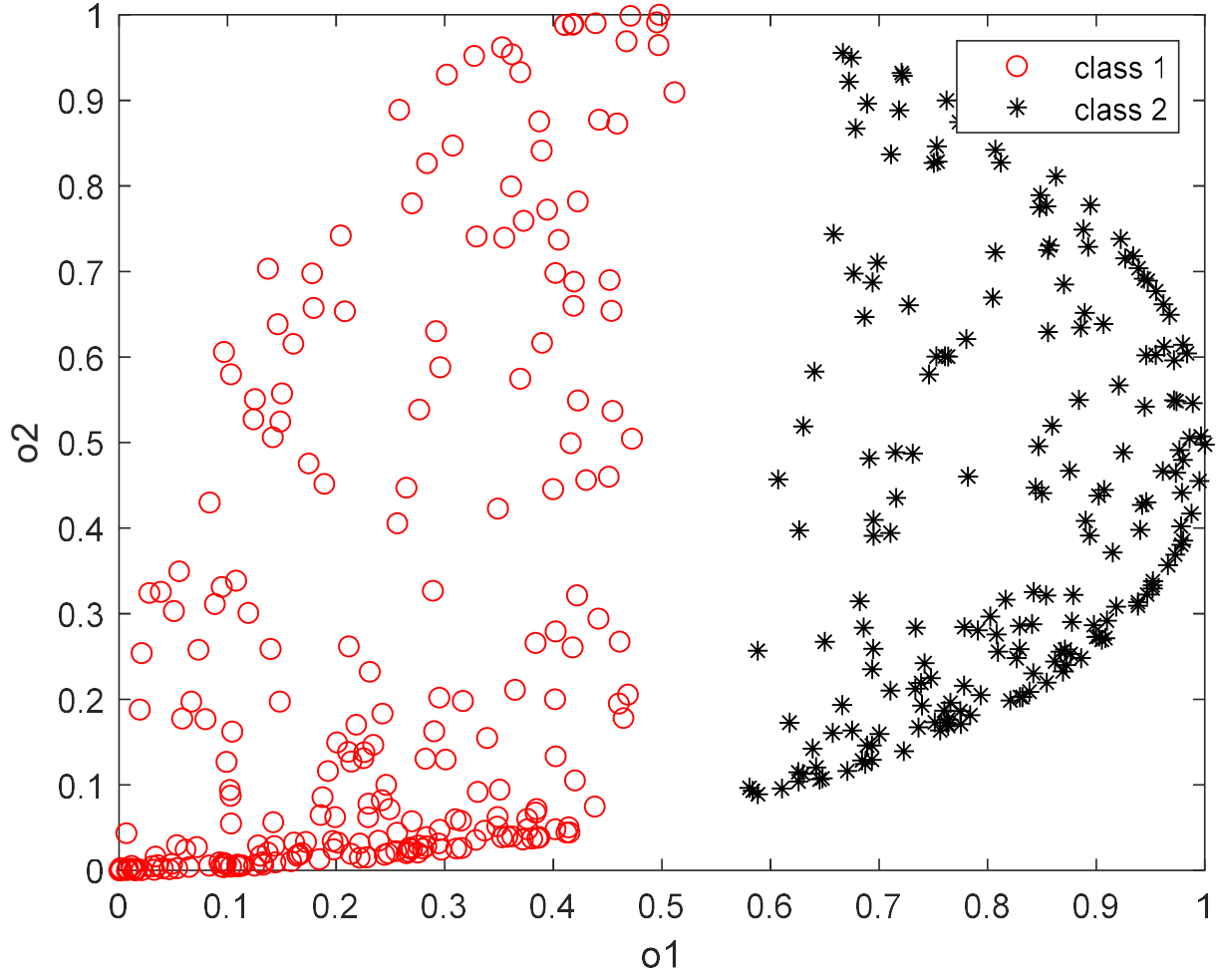


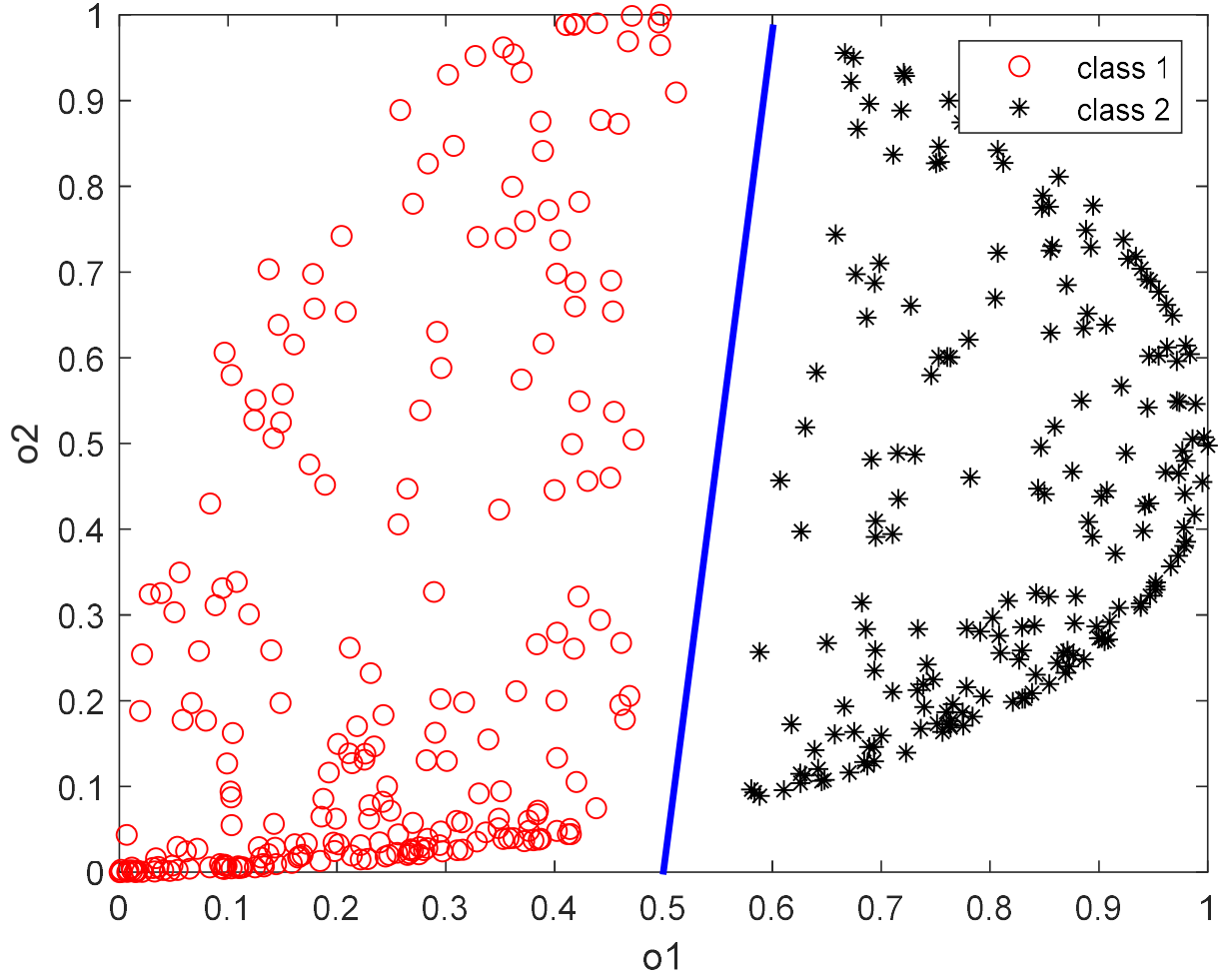


Nonlinear mapping Linear mapping


class 1
class 2
x1
x2
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3
4


class 1
class 2
x1
x2
4
3
2
1
0
-1
-2
-3
-4
-3
-2
-1
0
1
2
3
4


class 1
class 2
o1
o2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


class 1
class 2
o1
o2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

The linear weights associated with the output neuron of the network and the parameters of the hidden layer neurons may be learned separately:

(1) In the first step, the RBF centers are determined.

(2) In the second step, the weights are estimated using error-correction learning, with the goal of minimizing the difference between the desired output and the actual output of the RBF neural network.

Weight estimation

It is noted that the output of the RBF neural network has a linear relationship with the output of hidden layer neurons, thus the estimation of weights in the second step can be done using a linear least square estimation algorithm.

Assume there are N training samples:

$$\{\mathbf{x}(1), d(1)\}, \{\mathbf{x}(2), d(2)\},..., \{\mathbf{x}(N), d(N)\}$$

Where d(k) is the target value (or class label in tasks of pattern classification) of **x**(k).

We now assume the centre of each neuron at the hidden layer is known, thus, for input vector **x**(k), we have:

$$f[\mathbf{x}(k)] = \sum_{j=1}^{m} w_j \exp\left(-\frac{\left\| \mathbf{x}(k) - \mathbf{c}_j \right\|^2}{2\sigma^2} \right) = \sum_{j=1}^{m} w_j o_j(k)$$

If each of the N sample is input to the RBF neural network, we can obtain N equations. Summarizing the N equations in a matrix form, yields:

$$\begin{bmatrix} f[\mathbf{x}(1)] \\ f[\mathbf{x}(2)] \\ \vdots \\ f[\mathbf{x}(N)] \end{bmatrix} = \begin{bmatrix} o_1(1) & o_2(1) & \cdots & o_m(1) \\ o_1(2) & o_2(2) & \cdots & o_m(2) \\ \vdots & \vdots & \ddots & \vdots \\ o_1(N) & o_2(N) & \cdots & o_m(N) \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

Define:

$$\mathbf{f} = \begin{bmatrix} f[\mathbf{x}(1)] \\ f[\mathbf{x}(2)] \\ \vdots \\ f[\mathbf{x}(N)] \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{\Phi} = \begin{bmatrix} o_1(1) & o_2(1) & \cdots & o_m(1) \\ o_1(2) & o_2(2) & \cdots & o_m(2) \\ \vdots & \vdots & \ddots & \vdots \\ o_1(N) & o_2(N) & \cdots & o_m(N) \end{bmatrix}$$

We obtain:

$$\mathbf{f} = \mathbf{\Phi w}$$

We need to find such weights that the following cost function is minimized:

$$J = \sum_{k=1}^{N} \{ f[x(k)] - d(k) \}^2$$

$$= [\mathbf{f} - \mathbf{d}]^{\mathbf{T}} [\mathbf{f} - \mathbf{d}]$$

$$= [\mathbf{\Phi w} - \mathbf{d}]^{T} [\mathbf{\Phi w} - \mathbf{d}]$$

Where

$$\mathbf{d} = \begin{bmatrix} d(1) & d(2) & \cdots & d(N) \end{bmatrix}^T$$

Solve

$$\frac{\partial J}{\partial \mathbf{w}} = 0$$

We obtain:

$$\mathbf{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{d}$$

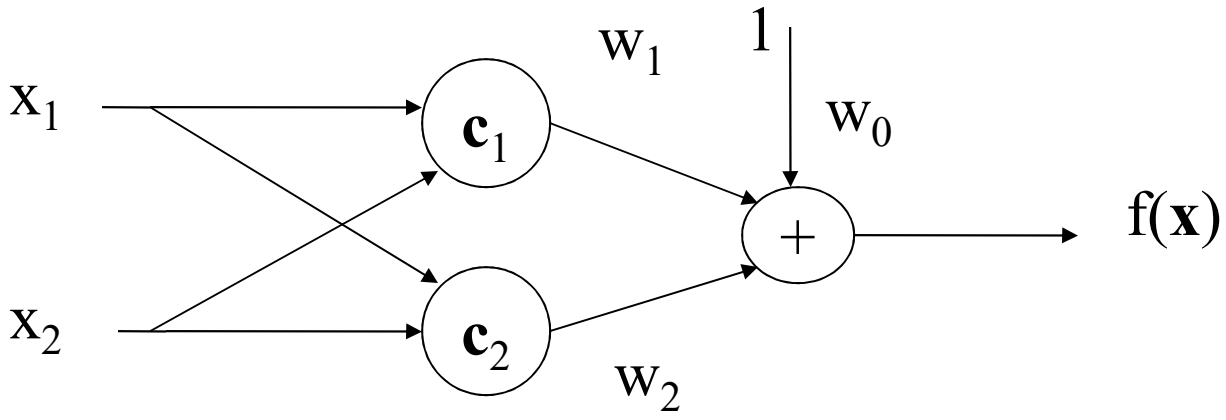
The weights thus obtained are called linear least square estimates.

Example

Consider the exclusive OR (XOR) problem again. The input and output of the logic operator are as follows:

Index of the data	inputs	outputs
1	$[1\ 1]^T$	0
2	$[0\ 1]^T$	1
3	$[0\ 0]^T$	0
4	$[1\ 0]^T$	1

Assume two hidden layer neurons are used, the centers of two neurons are set to: $\mathbf{c}_1=[1\ 1]^T$, $\mathbf{c}_2=[0\ 0]^T$, and the width of the Gaussian basis function is set to 0.707.



$$f(\mathbf{x}) = \sum_{i=1}^{2} w_i \exp(-\| \mathbf{x} - \mathbf{c}_i \|^2 / 2\sigma^2) + w_0$$

Substituting all the 4 data points to the above equation, we obtain the following matrix equation:

$$\begin{bmatrix} f[\mathbf{x}(1)] \\ f[\mathbf{x}(2)] \\ f[\mathbf{x}(3)] \\ f[\mathbf{x}(4)] \end{bmatrix} = \begin{bmatrix} 1 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} = \mathbf{\Phi w}$$

Define

$$\mathbf{d} = \begin{bmatrix} d(1) \\ d(2) \\ d(3) \\ d(4) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Then the weight estimated are obtained as follows:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{d}$$

$$= \left(\begin{bmatrix} 1 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \end{bmatrix}^T \begin{bmatrix} 1 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \end{bmatrix}^T \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} -2.5018 \\ -2.5018 \\ +2.8404 \end{bmatrix}$$

Let's check the performance of the RBF neural network.

If $\mathbf{x}=[\ 1\ 1]^T$, we have $f(\mathbf{x})=0.000106$

If $\mathbf{x}=[\ 0\ 1]^T$, we have $f(\mathbf{x})=1.0001$

If $\mathbf{x}=[\ 0\ 0]^T$, we have $f(\mathbf{x})=0.000106$

If $\mathbf{x}=[\ 1\ 0]^T$, we have $f(\mathbf{x})=1.0001$

Obviously the RBF neural network with just two neurons could approximate the XOR logic operation perfectly well.

Strategies for neuron center determination

There are a few strategies that we can follow in the determination of hidden layer neuron centers. A few example strategies are introduced next.

Random selection from training samples

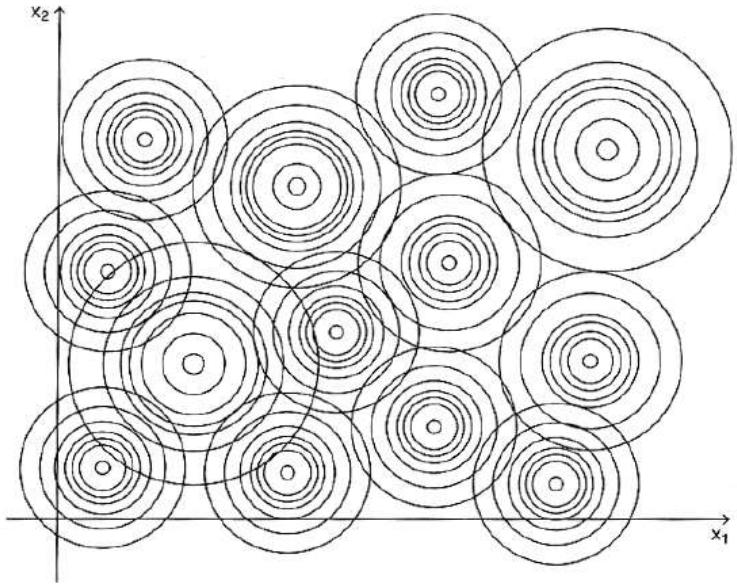
This is the simplest approach. The locations of the centers are chosen randomly from the training data. This is considered to be a sensible approach provided that the randomly selected training sample are distributed in a representative manner for the training data.

For the radial basis function, we may employ Gaussian function whose standard deviation is fixed according to the spread of the centers:

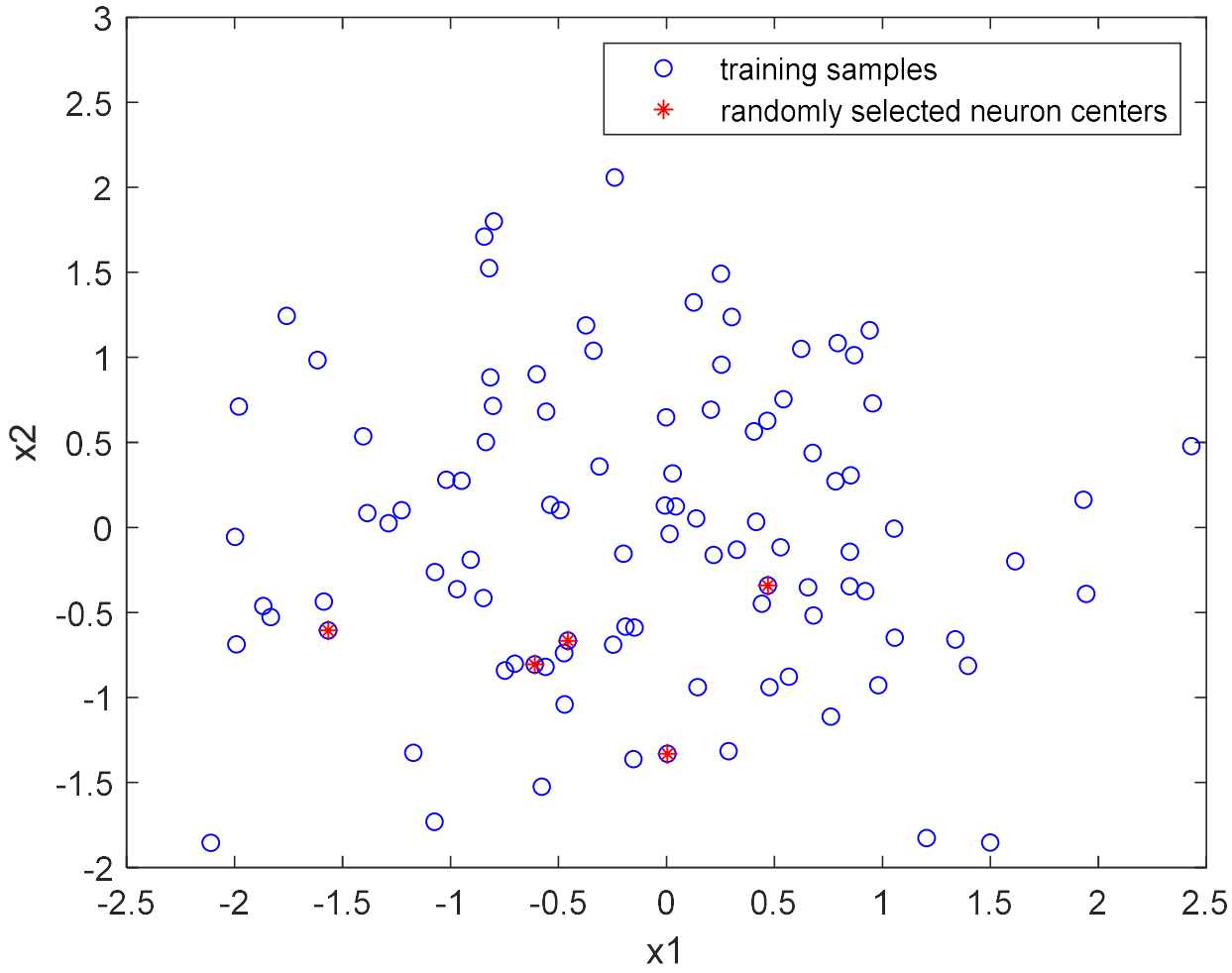
$$\sigma = d_{max} / \sqrt{2m}$$

where d_{max} is the maximum distance between the chosen centers, and m is the number of centers. The above formula ensures that individual radial-basis functions are not too peaked or too flat (both of the two extreme conditions should be avoided).

As an alternative for basis function width determination, we may use individually scaled centers with broader width in areas of low density, and narrower width in areas of high density.



Random selection method is easy to implement, but this method cannot guarantee a good coverage of the input space if a small number of neurons are used as shown below, where 5 samples are randomly chosen.



Why do we need a good coverage of the input space?

Prototypes of training samples as neuron centres

The randomly selected centers cannot guarantee good coverage of the input space. Then a natural way of solving this problem is to select and use prototypes of training samples as neuron centers. In this approach, the basis functions are permitted to place the centers of the RBF functions in only those regions of the input space where significant data are presented. SOM neural networks can be used to fulfill this task.

Besides SOM neural network, other clustering algorithms such as K-means clustering can also be used to select prototypes of the training samples.

The K-means clustering algorithm is essentially an iterative procedure, whose details are described below:

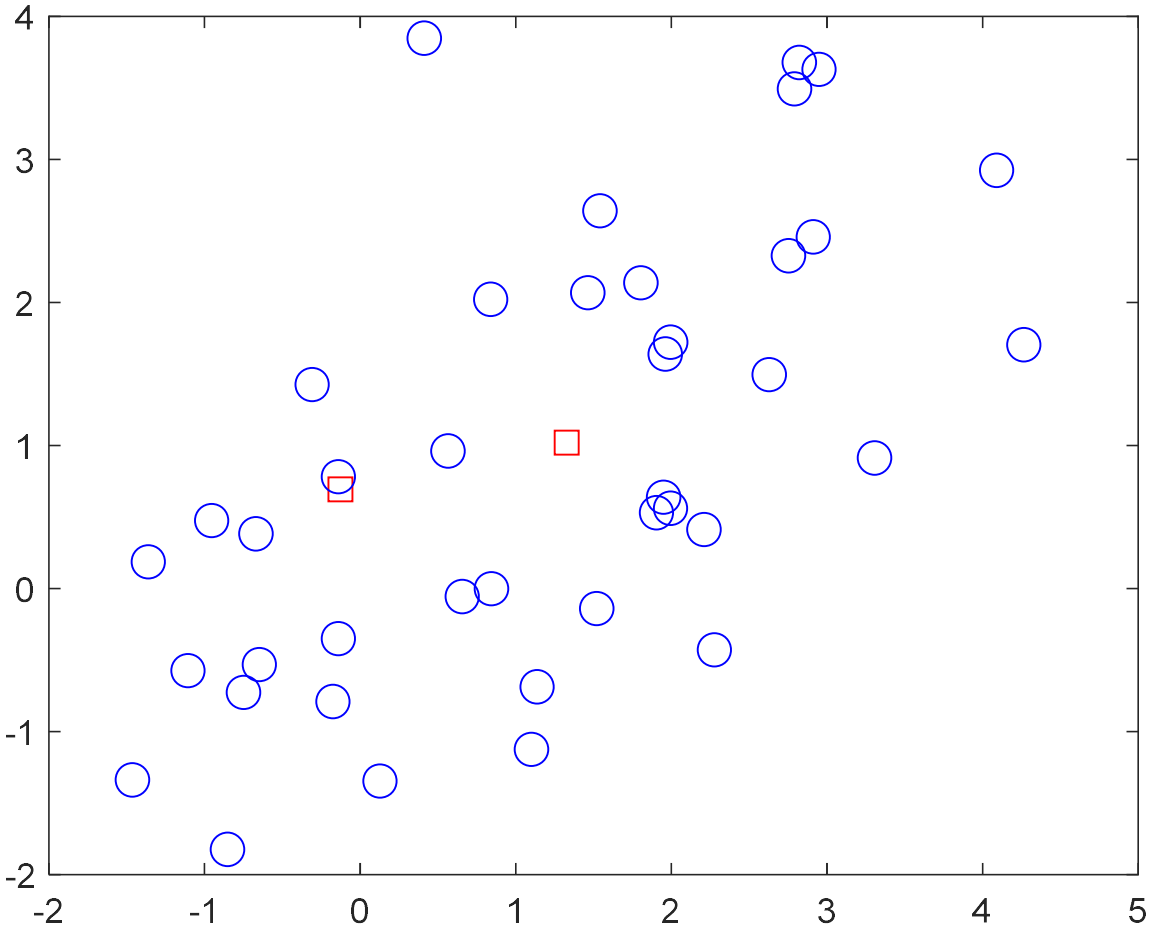
K-means clustering algorithm

(a) Initialize cluster center positions either by randomly selecting from the samples or by some prior knowledge.

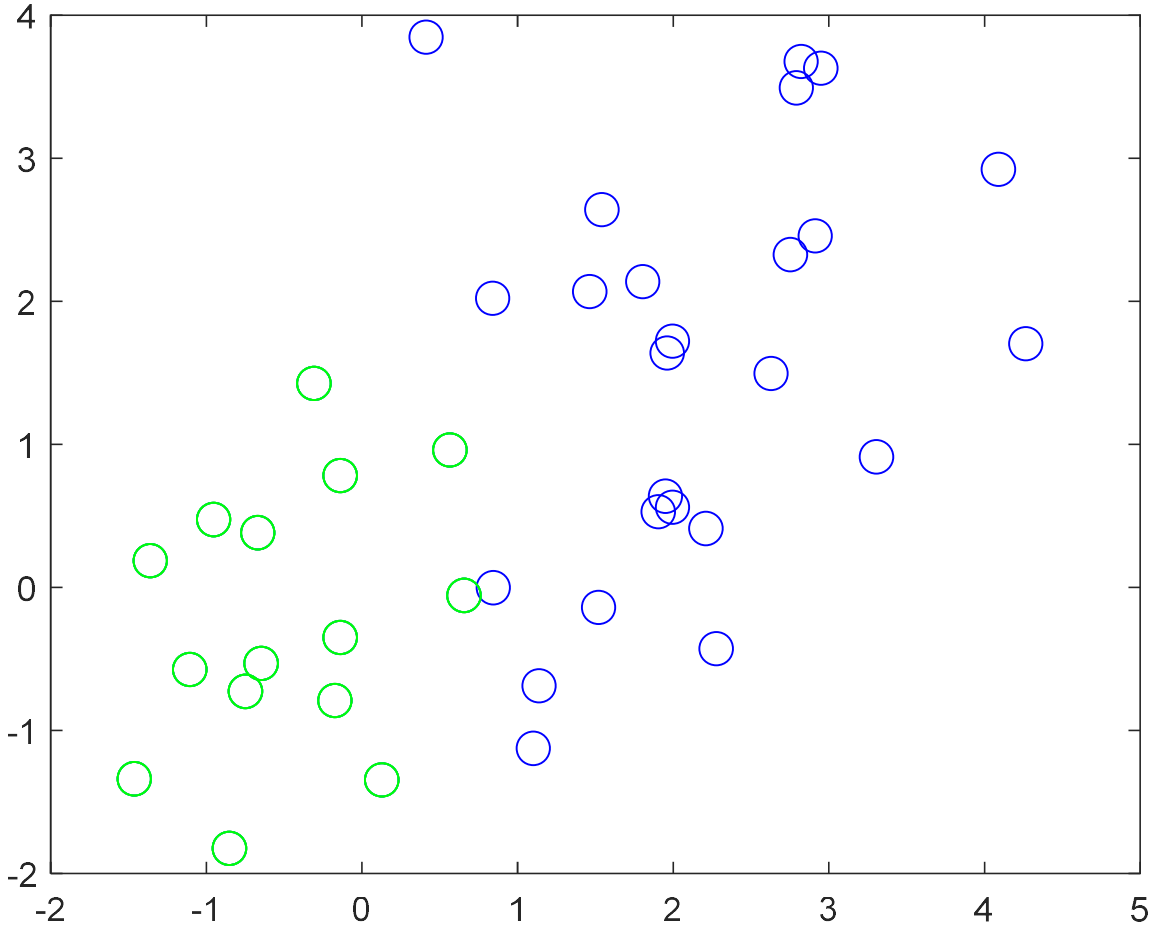
(b) Perform partitioning by assigning samples to the nearest clusters. This is usually done by first calculating the distance between each sample to each cluster center and then assigning samples to the cluster with the smallest distance.

(c) Find new cluster centers by averaging the samples in the same cluster.

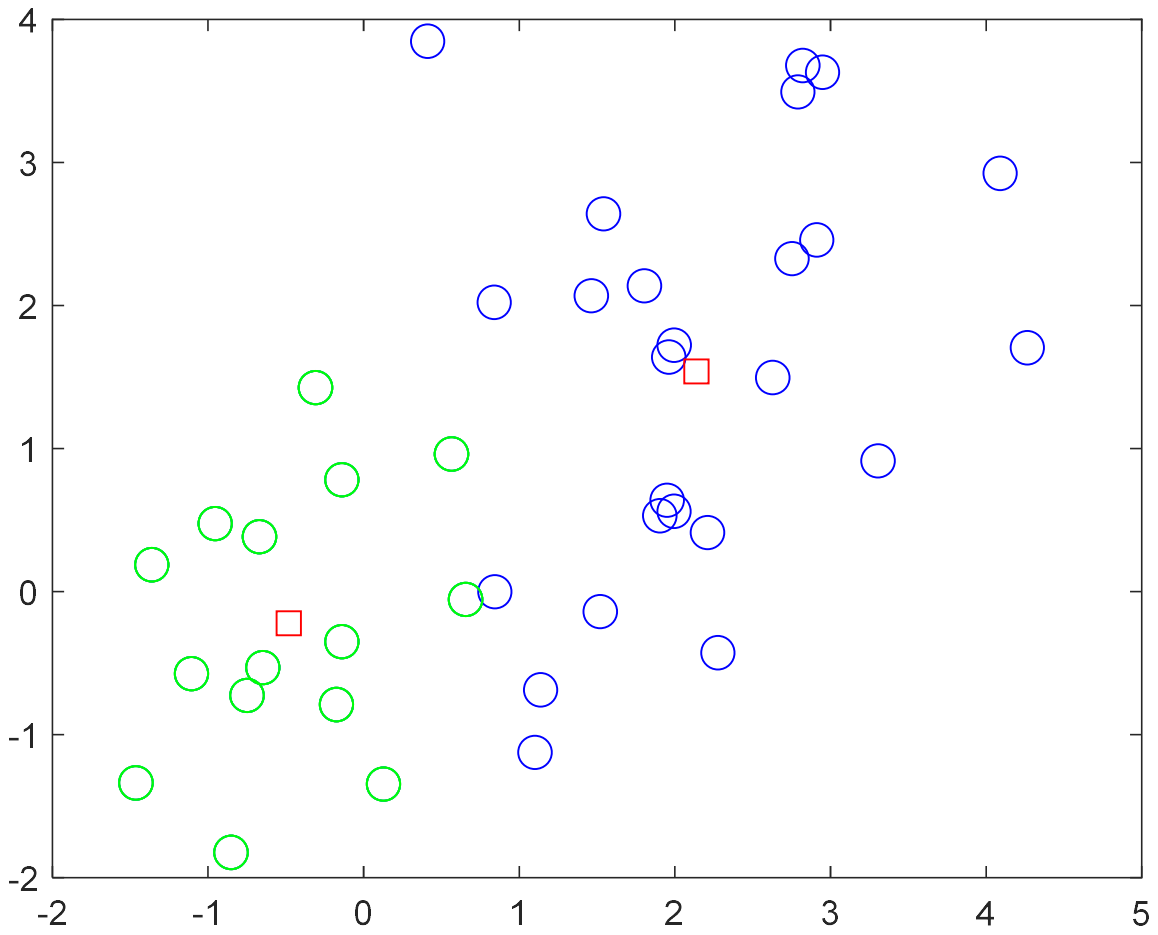
(d) Go to step (b) until the cluster center positions and the sample partitioning are no longer changed.



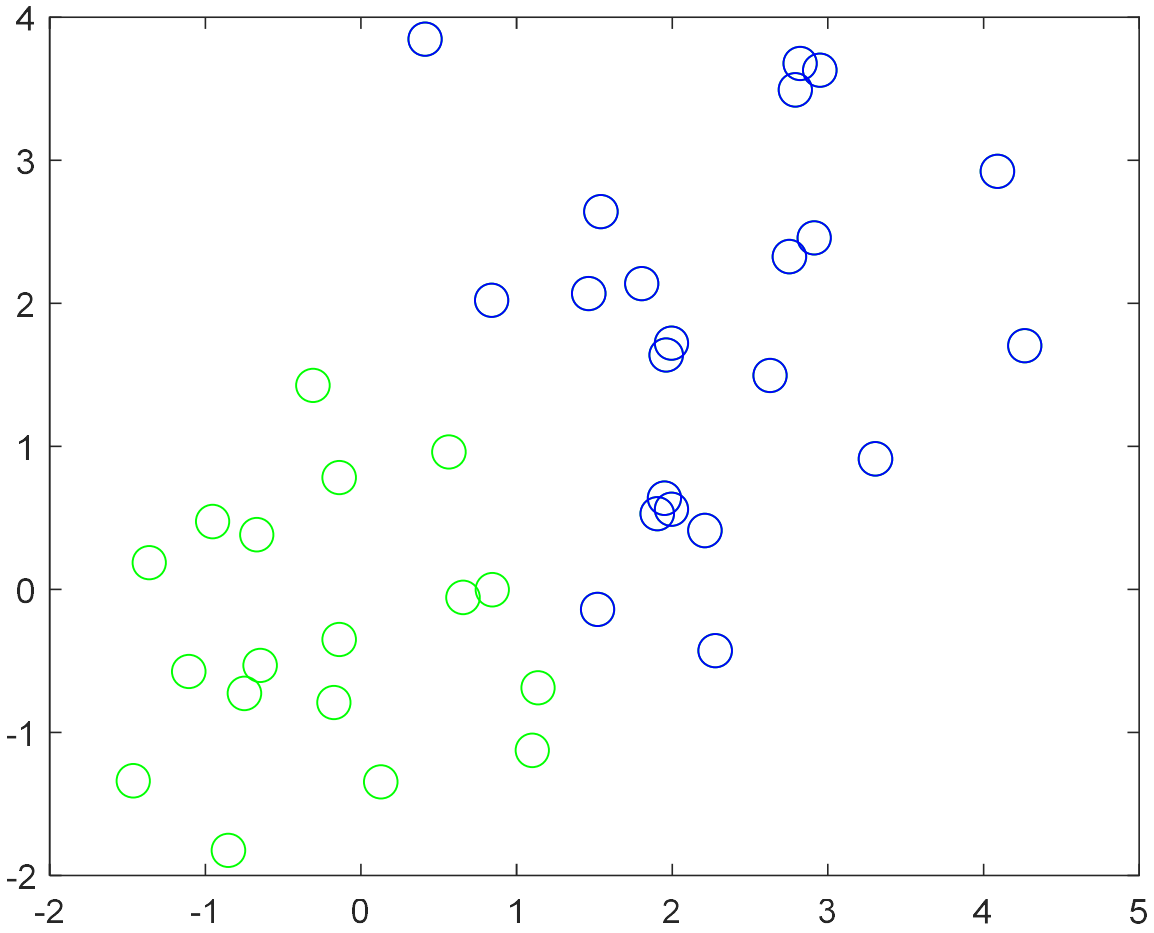
(a) Initialization of cluster centers



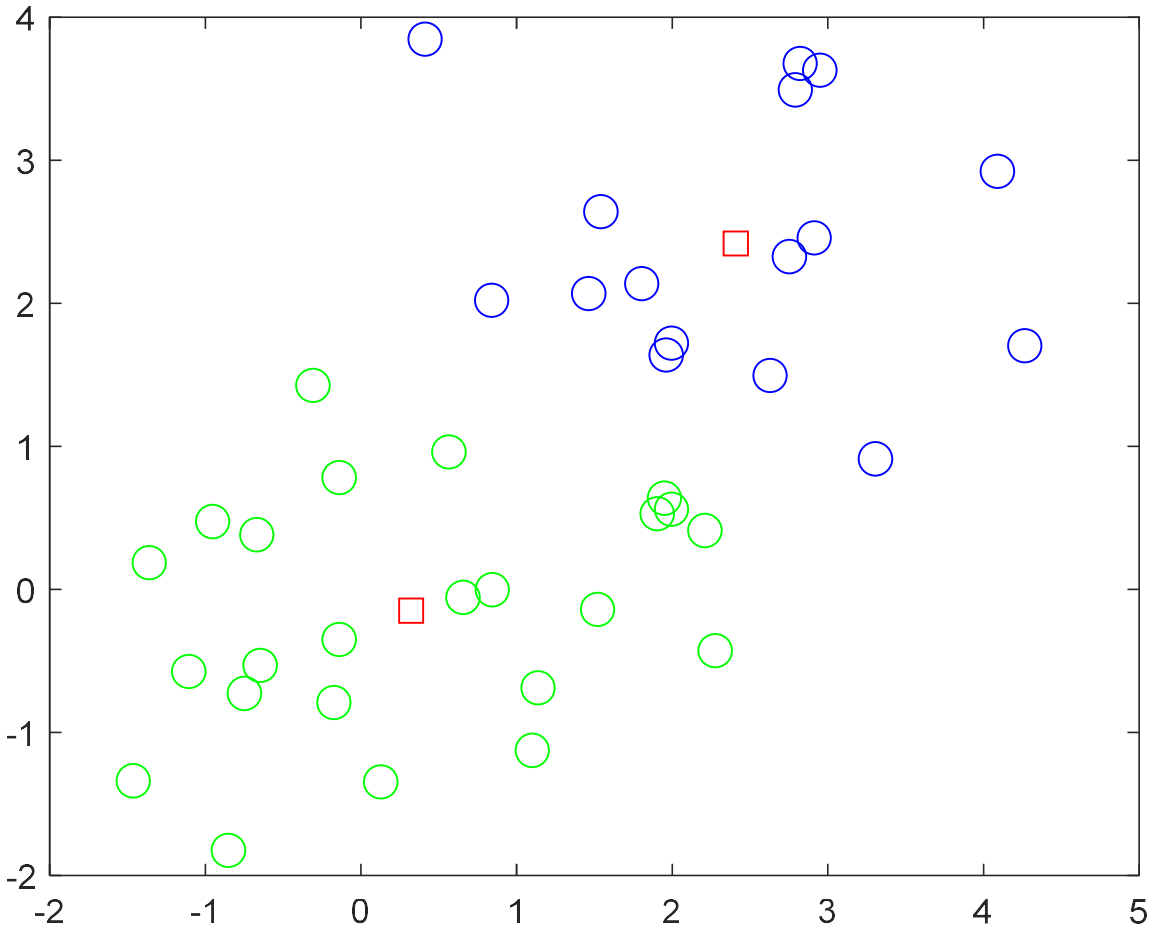
(b) Clustering



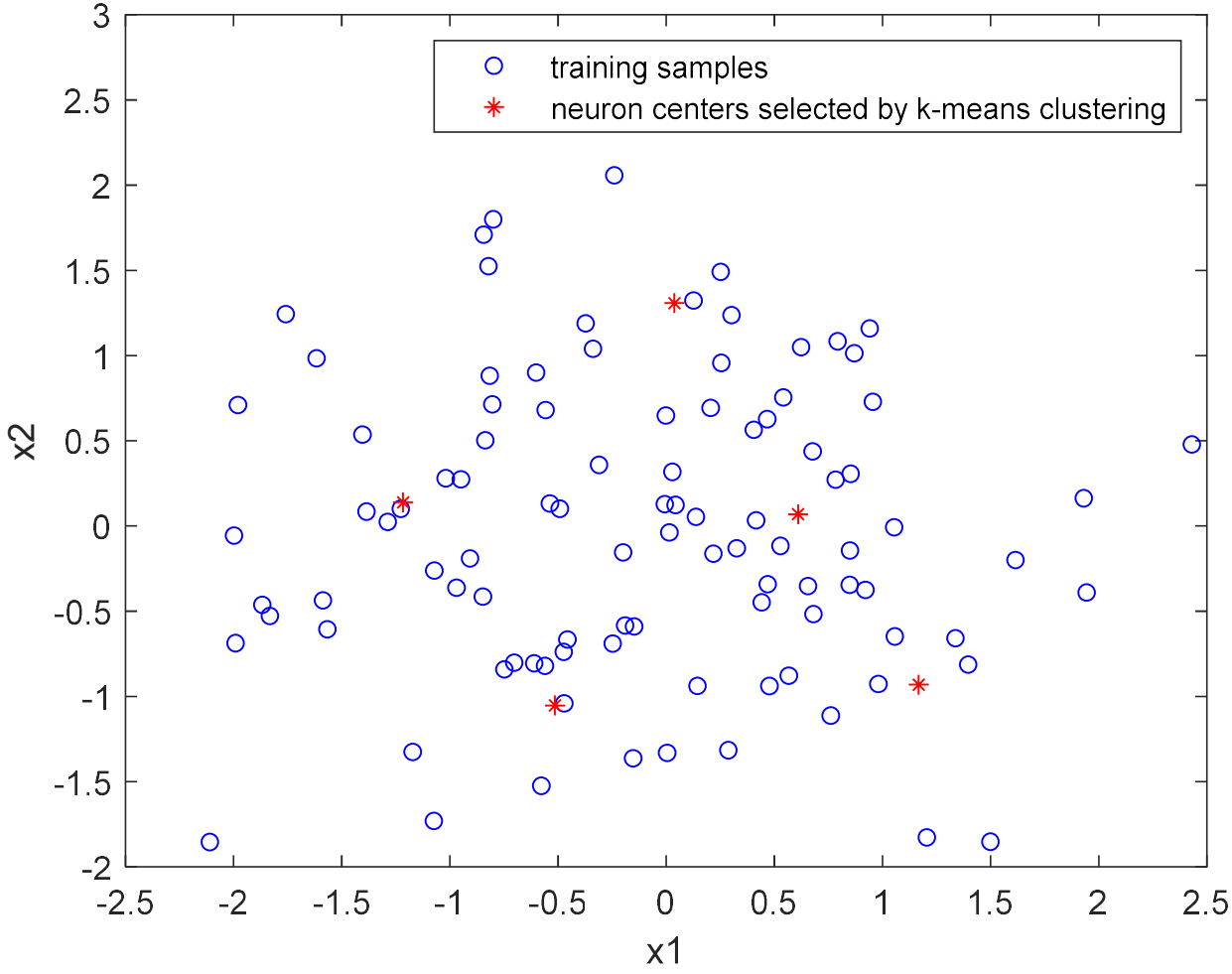
(c) Finding new cluster centers



(d) Clustering based on new centers.



(e) Final clustering results


training samples
neuron centers selected by k-means clustering
x1
x2
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
2.5

Center selection as a model selection problem

What is model selection?

Model selection is a well studied problem in statistics. Assuming that we have a set of candidate variables:

$$\{x_1, x_2, \cdots, x_n\}$$

Model selection aims to determine what variables should be included in the regression model:

$$y = a_1 z_1 + a_2 z_2 + \cdots + a_m z_m$$

$$z_j \in \{x_1, x_2, \cdots, x_n\}$$

$$m < n$$

For RBF neural network with m neurons, we have:

$$f(\mathbf{x}) = \sum_{j=1}^{m} w_j \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2\sigma^2}\right) = \sum_{j=1}^{m} w_j o_j(\mathbf{x})$$

f(**x**) is a nonlinear function of **x**, but it has a linear relationship with the o_j, which are the output of hidden layer neurons. If we consider each of the available training sample $\mathbf{x}_i$, i=1,2, N, as a candidate neuron centre, then we will have N available variables:

$$\{o_1, o_2, \cdots, o_N\}$$

Selecting $\boldsymbol{c}_j$ becomes the problem of selecting o_j to include in the network:

$$f(\mathbf{x}) = w_1 z_1 + w_2 z_2 + \cdots + w_m z_m$$

$$z_j \in \{o_1, o_2, \cdots, o_N\}$$

Typical strategies in model selection:

There are two typical strategies in model selection, namely top-down and bottom-up.

Top down:

In the top down strategy, we start with a full model and then remove the variables that do not contribute significantly to the performance of the model. The sequential backward elimination method belongs to this strategy.

$$y = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \qquad \text{full model}$$

$$y = a_1 z_1 + a_2 z_2 + \cdots + a_{n-1} z_{n-1} \qquad \text{reduced model}$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$y = a_1 z_1 + a_2 z_2 + \cdots + a_m z_m \qquad \text{final model}$$

Bottom-up:

The bottom up strategy starts with an empty model, then slowly adds potential variables. The sequential forward selection method belongs to this strategy.

$$y = a_1 z_1$$

$$y = a_1 z_1 + a_2 z_2$$

$$\vdots$$

$$y = a_1 z_1 + a_2 z_2 + \cdots + a_m z_m$$

where

$$z_j \in \{x_1, x_2, \cdots, x_n\}$$

The addition or removal of a neuron to or from the network is based on the contribution of the neuron to the performance of the network. The contribution of a neuron is often evaluated based on the approximation error reduction or inflation after including or removing the neuron.

The approximation error is defined as:

$$E = \frac{1}{2}\sum_{k=1}^{N} e_k^2$$

where N is the total number of training samples, e_k is the error signal between the desired output d(k) and the output of the network:

$$e_k = f[\mathbf{x}(k)] - d(k)$$

Approximation error-based sequential forward selection algorithm is described next. This algorithm is an iterative procedure that builds up the network step by step.

Sequential forward selection algorithm for RBF neural network

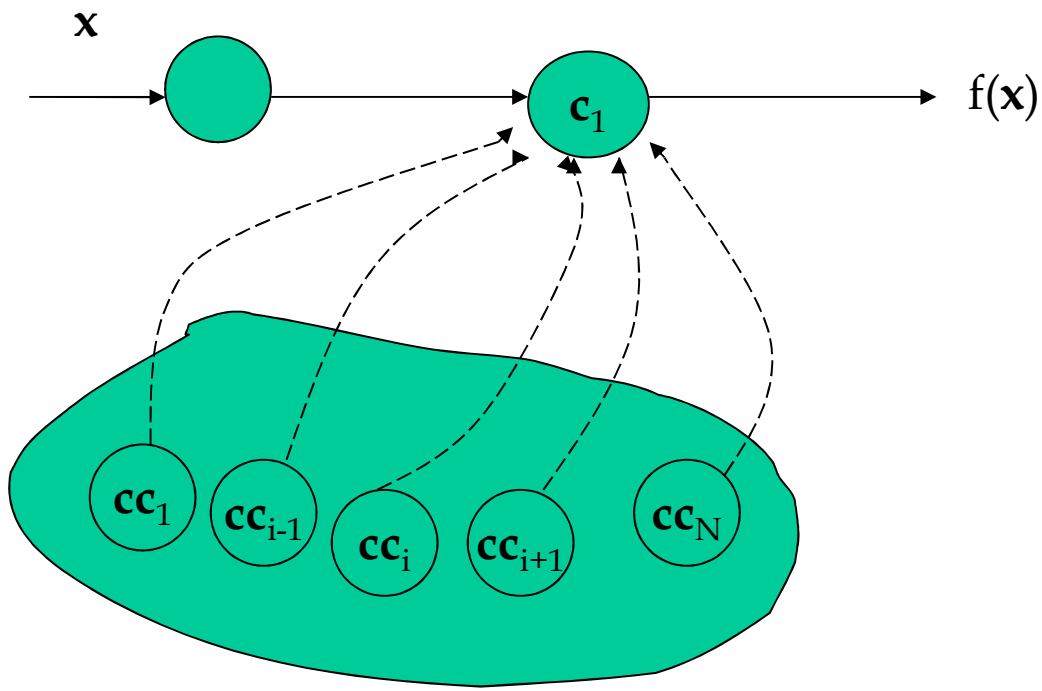
(1) In the first step, consider all training samples (vectors) as the candidate center of the first neuron to be selected:

$$\mathbf{c}_1^j = \mathbf{x}_j \qquad j=1,2,\ldots N$$

Construct N neural networks with 1 hidden neuron using the N candidate neurons respectively, estimate the weight and calculate the approximation error. The neuron that leads to minimum approximation error is selected as the first neuron $\mathbf{c}_1$.

At Step 1:

Consider each sample as a center of the neurons to construct a 1-neuron RBF neural network;



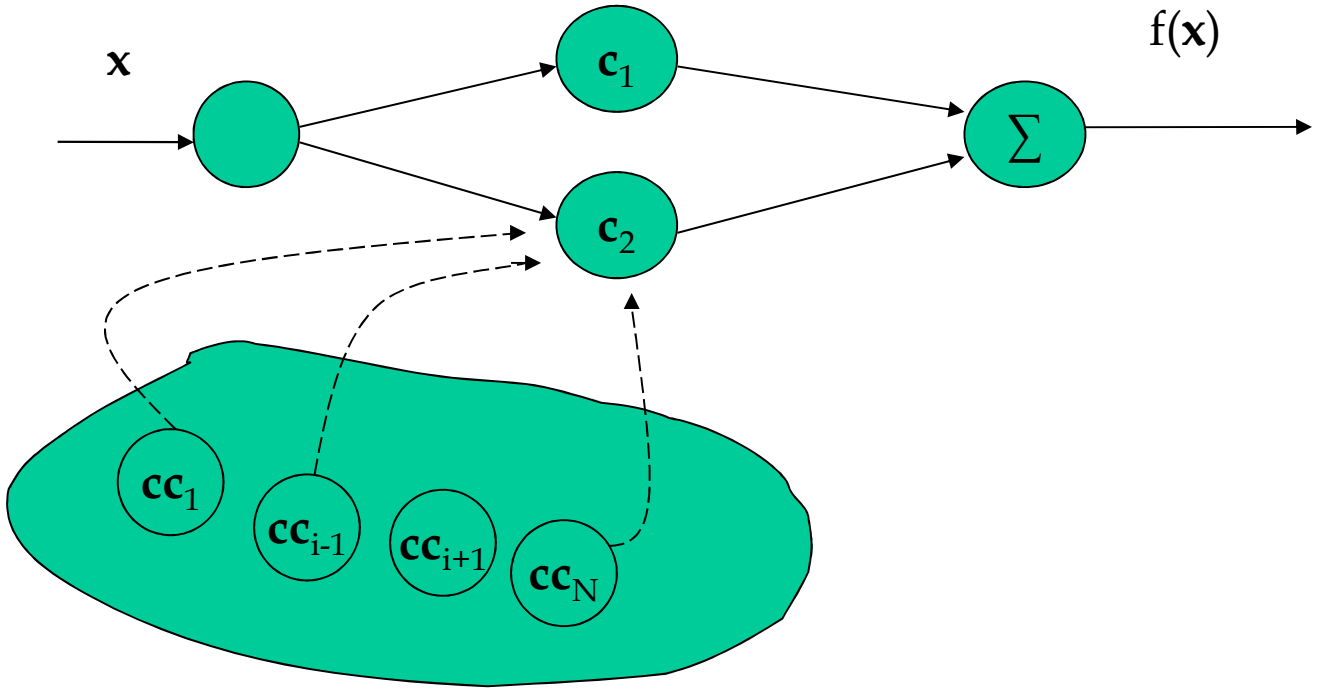
Pool of candidate neuron centers

(2) At the second step, consider each of the remaining (N-1) samples as a candidate of center to be secondly selected, construct N-1 2-neuron neural networks using each of the candidate center together with the selected neuron center c_1.

Estimate the weights, calculate the approximation error. The candidate that leads to the minimum approximation error is selected as the second center.

At Step 2:

Consider each neuron center in the pool as the one to be selected next, and construct 2-neuron RBF neural networks.



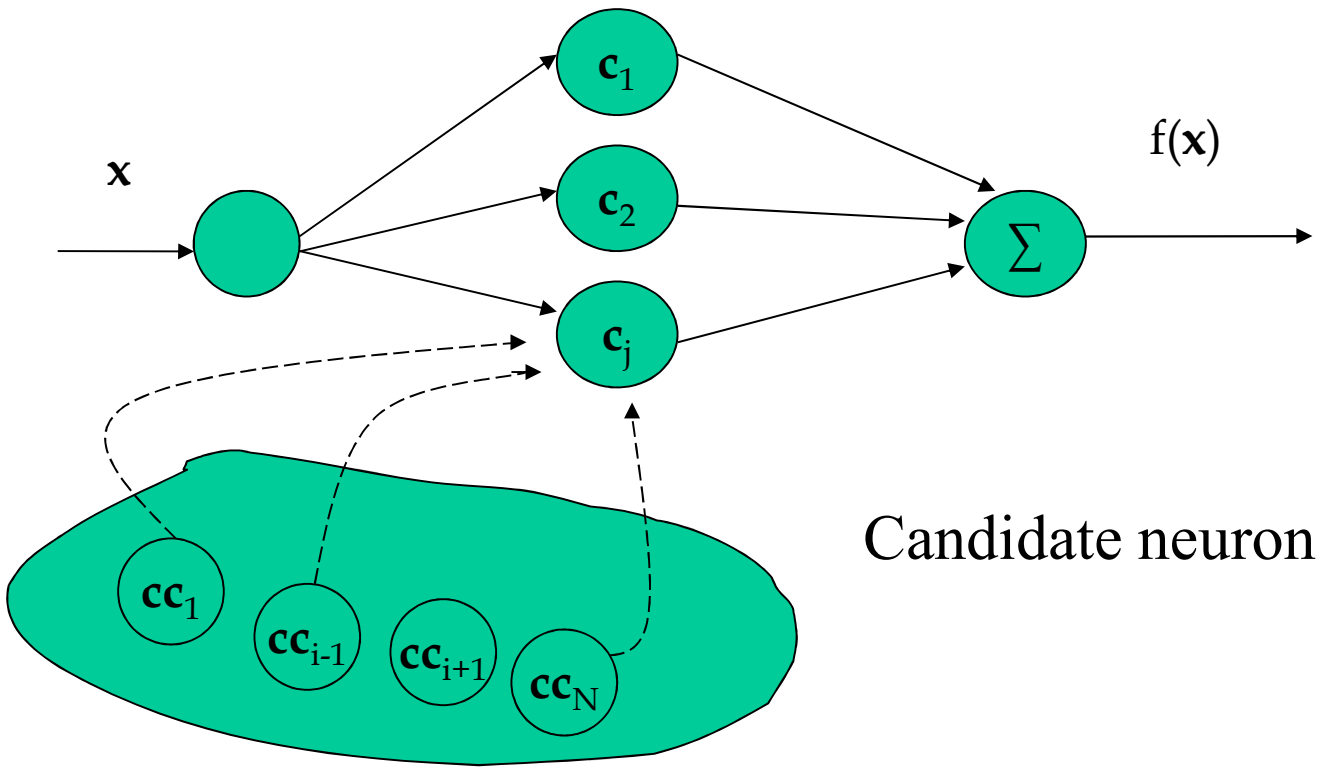
Pool of candidate neuron centers

(3) The above procedure is continued until the stopping criterion is satisfied. The stopping criterion can be:

1) pre-defined number of neurons;

2) Pre-defined accuracy.

At Step 3:



The above processing until the stopping criterion is satisfied.

Optimization-based learning (weights and centers)

In this approach, the centers of the RBF and all other parameters of the network undergo a single learning process. A natural candidate for such a process is error-correction learning, which is most conveniently implemented using a gradient-descent procedure.

The first step in such a procedure is to define the criterion to be optimized:

$$E = \frac{1}{2}\sum_{k=1}^{N} e_k^2$$

where N is the number of training samples, e_j is the error signal between the desired output d(j) and the network output f[**x**(k)]:

$$e_k = f[\mathbf{x}(k)] - d(k)$$

By minimizing the cost function E using gradient-descent, we can obtain all parameters of the RBF neural network:

$$\mathbf{c}_j, w_j, \sigma = \arg\min(E)$$

The gradient-descent based optimization can be summarized as follows:

(1) Weight estimation:

$$\frac{\partial E(n)}{\partial w_j(n)} = \sum_{k=1}^{N} e_k(n) \exp\left(- \frac{\left\| \mathbf{x}(k) - \mathbf{c}_j(n) \right\|^2}{2\sigma^2} \right)$$

$$w_j(n+1) = w_j(n) - \eta_1 \frac{\partial E(n)}{\partial w_j(n)}$$

(2) Center location estimation

$$\frac{\partial E(n)}{\partial \mathbf{c}_j(n)} = w_j(n)\sum_{k=1}^{N} e_k(n)\exp\left(-\frac{\left\|\mathbf{x}(k)-\mathbf{c}_j(n)\right\|^2}{2\sigma^2}\right)\frac{\mathbf{x}(k)-\mathbf{c}_j(n)}{\sigma^2(n)}$$

$$\mathbf{c}_j(n+1) = \mathbf{c}_j(n) - \eta_2 \frac{\partial E(n)}{\partial \mathbf{c}_j(n)}$$

(3) Width estimation

$$\frac{\partial E(n)}{\partial \sigma(n)} = \sum_{k=1}^{N} e_k(n)\sum_{j=1}^{m} w_j \exp\left(-\frac{\left\|\mathbf{x}(k)-\mathbf{c}_j(n)\right\|^2}{2\sigma^2}\right)\frac{\left\|\mathbf{x}(k)-\mathbf{c}_j(n)\right\|^2}{\sigma^3(n)}$$

$$\sigma(\mathrm{n}+1) = \sigma(\mathrm{n}) - \eta_3 \frac{\partial \mathrm{E}(\mathrm{n})}{\partial \sigma(\mathrm{n})}$$

where η_1, η_2, and η_3 are step-size. The iteration is repeated until parameters have no noticeable changes between two iterations.

Application example of RBF neural networks in data mining

Direct mailings to a potential customers can be a very effective way to market a product or a services. However, as we all know, many of this junk mails are of no interest to the people to receive it. And most of the mails are thrown away. This wastes and costs money.

If the company has a better understanding of who their potential customers are, they can know more accurately who to send mails. Thus, the cost can be reduced.

We can understand customers by analyzing their personal data:

(1) Select a small groups of customers of various background, and collect their responds (training data).

(2) Train a pattern classifier using the above small group of data.

(3) Predict the interest of other customers and select those that have high possibility to be interested.

An insurance company wishes to launch a caravan insurance plan. The problem under study is to predict which customers are potentially interested in the caravan insurance policy.

(1) Training data set.

The training data set contains 5822 customers. The record of each customer consists of 85 attributes (features/variables) including socio-demographic data and product ownership. The socio-demographic data is derived from zip-codes. In the training data, the information of whether or not the customers have a caravan policy (class labels) is also known.

Among the 5822 customers, only 348 customers have a caravan policy.

(2) Test data set

The test data contains 4000 customers, of whom the information of whether they have a caravan policy is NOT included. The problem here is to select 800 customers that are most interested in the policy from the 4000 customers.

First, we need to train a classifier using the training data. Among the 5822 customers, 348 have a policy (class 1), while 5474 do not (class 2). The data exhibits an unbalanced problem (5474:348). So in the classifier training, we select 500 customers from the 5474 customers of class 2 using SOM neural networks.

Another problem with the training data is that, among the 85 attributes available, many are irrelevant, insignificant or redundant. So, before the classifier training, we need to perform attribute selection (feature selection) by eliminating the irrelevant, insignificant and redundant variables, and keeping the useful attributes.

The usefulness of an attribute subset can be evaluated based on its ability to provide large class separation. Based on this criterion, we select 12 attributes, which are used as input variables to the RBF neural network classifier.

After feature selection, we next determine the number of neurons at the hidden layer and the center locations for each neuron.

To select neuron center locations, we first consider each of the 848 (348+500) training samples as candidates, and then use the bottom-up method to select 7 neurons.

The weights that connect hidden layer neurons and the output layer neuron are estimated using a linear least algorithm which minimizes the following cost function:

$$J = \sum_{k=1}^{848} \{ f[\mathbf{x}(k)] - d(k) \}^2$$

where d(k) is the class label of the k^{th} training sample, having a value of 1 (class 1) or -1 (class 2). f[**x**(k)] is the network output for the k^{th} training sample.

With the RBF classifier learned, we select 800 customers that most possibly have a caravan insurance policy. The number of correctly identified policy owner is 105. Actually, there are 238 policy owners among the 4000 testing customers.

If we randomly select 800 customers from the 4000, averagely, the correctly identified policy owner would be:

$$\frac{800}{4000} \times 238 \approx 46$$

The advantage of data mining is obvious.