Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 14η

Radial Basis Function (Neural) Networks

Radial Basis Network basics

- The radial basis network is a two-layer network. There are two major distinctions between the *radial basis function* (RBF) *network* and a two layer perceptron:
  1. In layer 1 of the RBF network, instead of performing an inner product operation between the weights and the input (matrix multiplication), we calculate the distance between the input vector and the rows of the weight matrix
  2. Second, instead of adding the bias, we multiply by the bias. Therefore, the net input for neuron i in the first layer is calculated as follows:

$$n_i^1 = ||\mathbf{p} - {}_i\mathbf{w}^1||b_i^1$$

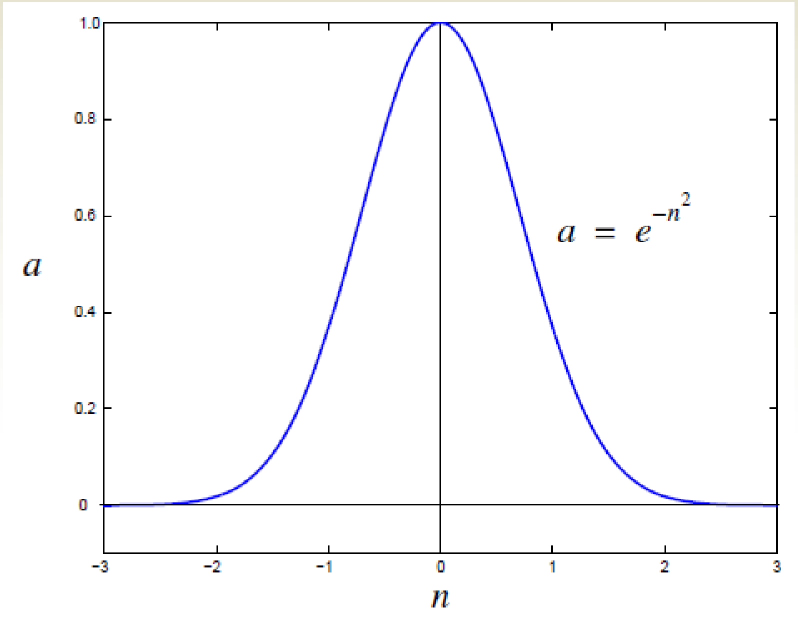
  - Each row of the weight matrix acts as a center point - a point where the net input value will be zero
  - The bias (std dev OR variance OR spread const) performs a scaling operation on the transfer (basis) function, causing it to stretch or compress

Radial Basis Network basics

- The transfer functions used in the first layer of the RBF network are different than the sigmoid functions (generally) used in the hidden layers of MLP
- There are several different types of transfer function:
  - D.S. Broomhead and D. Lowe, "*Multivariable function interpolation and adaptive networks*," **Complex Systems**, vol.2, pp. 321-355, 1988
- we will consider only the Gaussian function (the most commonly used in the neural network community). It is defined as follows:

$$\alpha = e^{-n^2}$$

A key property of this function is that it is *local* – the output is close to zero if you move very far in either direction from the center point. This is in contrast to the global sigmoid functions, whose output remains close to 1 as the net input goes to infinity



Other Radial Basis Functions

- A number of functions can be used as the RBF:

$$\phi(r) = e^{-r^2/2\sigma^2} \quad \text{(Gaussian)} \qquad \left[bias = \frac{1}{\sigma\sqrt{2}}\right]$$

$$\phi(r) = \frac{1}{(r^2+\sigma^2)^\alpha},\ a > 0$$

$$\phi(r) = (r^2+\sigma^2)^\beta,\ 0 < \beta < 1$$

$$\phi(r) = r \quad \text{(linear)}$$

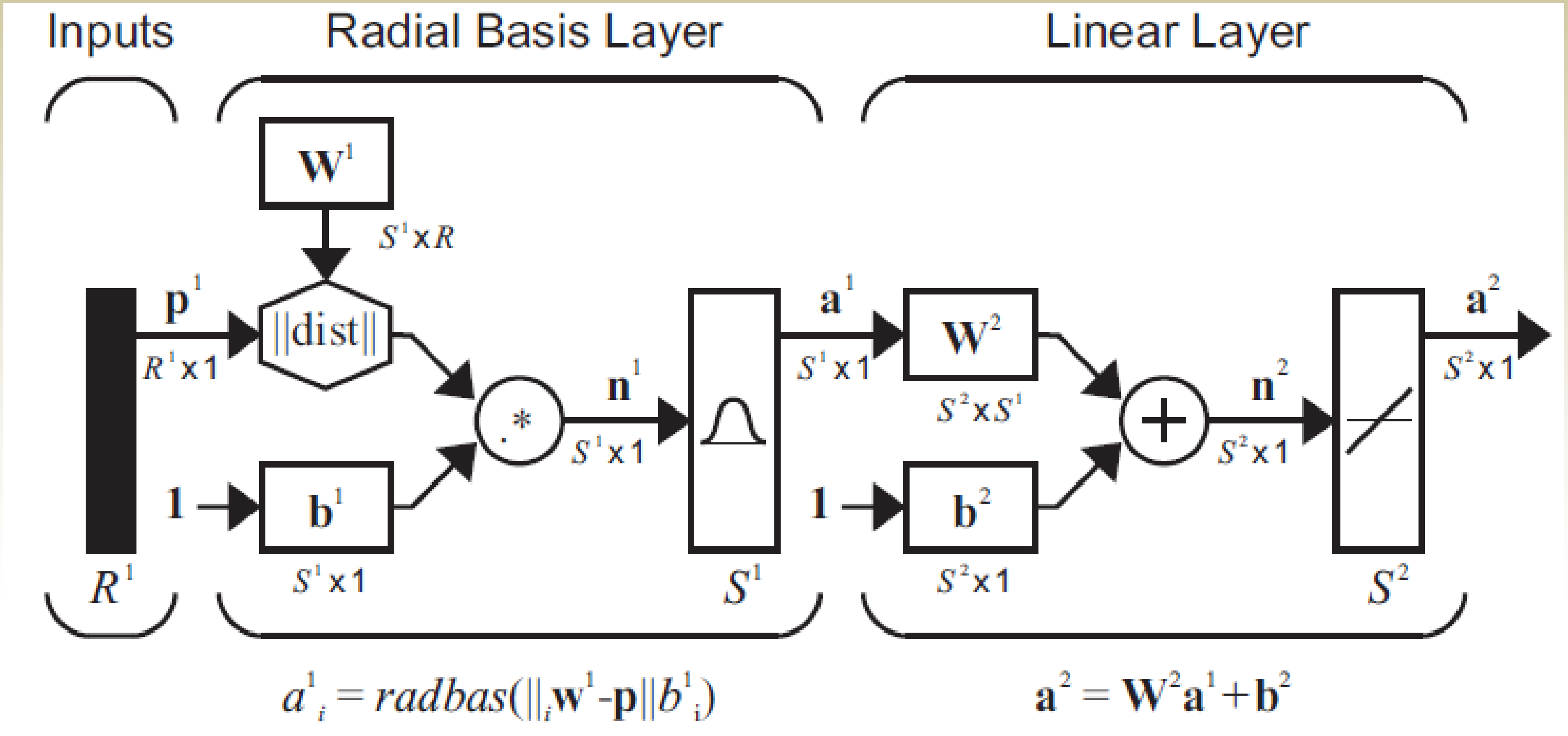
$$\phi(r) = r^2 ln(r) \quad \text{(thin-plate spline)}$$

$$\phi(r) = \frac{1}{1+e^{(r/sigma^2)-\theta}} \quad \text{(logistic function)}$$

Radial Basis Network basics

- The second layer of the RBF network is a standard linear layer:

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2$$



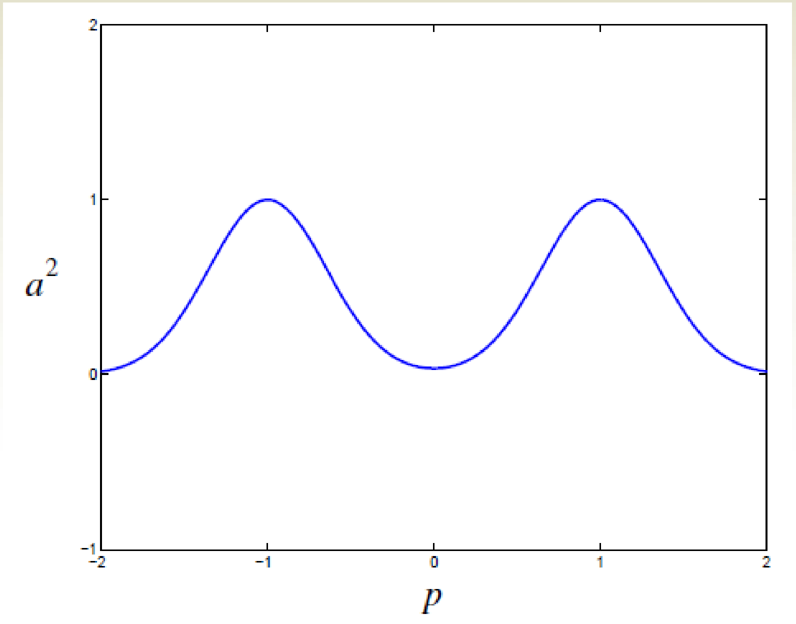
Function approximation example

- This RBF network has been shown to be a universal approximator (just like the MLP network)
  - J. Park and I.W. Sandberg, "*Universal approximation using radial-basis-function networks*," ***Neural Computation***, vol. 5, pp. 305-316, 1993.
- Consider a network with two neurons in the hidden layer, one output neuron, and with the following parameters:

$$w^1_{1,1} = -1, \quad w^1_{2,1} = 1, \quad b^1_1 = 2, \quad b^1_2 = 2$$

$$w^2_{1,1} = 1, \quad w^2_{1,2} = 1, \quad b^2 = 0$$

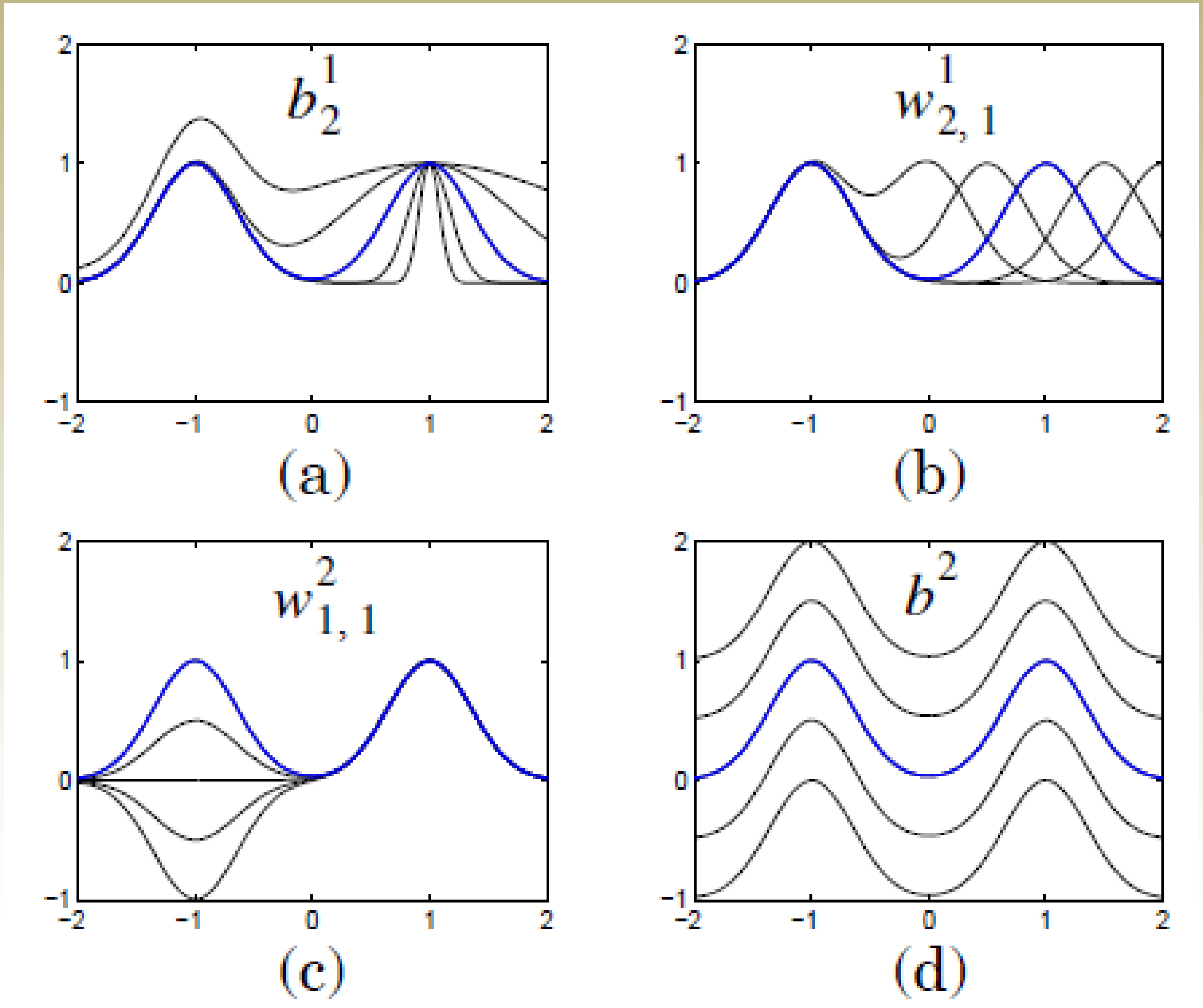
- The response consists of two hills, one for each of the Gaussian neurons (basis functions) in the first layer
- By adjusting the network parameters, we can change the shape and location of each hill



Function approximation example

Effects of parameter changes on response (The blue curve is the nominal response)

- the larger the bias, the narrower the hill
- the weights in the first layer determine the location of the hills (a hill centered at each first layer weight)
- the weights in the second layer scale the height of the hills
- The bias in the second layer shifts the entire network response up or down



Function approximation example

- If we have enough neurons in the first layer of the RBF network, we can approximate virtually any function of interest
- However, although both MLP and RBF networks are universal approximators, they perform their approximations in different ways
    - For the RBF network, each transfer function is only active over a small region of the input space - the response is local. If the input moves far from a given center, the output of the corresponding neuron will be close to zero
        - This has consequences for the design of RBF networks. We must have centers adequately distributed throughout the range of the network inputs, and we must select biases in such a way that all of the basis functions overlap in a significant way

Pattern classification example: the XOR

$$\text{Category 1: } \left\{ \mathbf{p}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \mathbf{p}_3 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right\}, \text{ Category 2: } \left\{ \mathbf{p}_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

- The idea is to have the network produce outputs greater than zero when the input is near patterns $\mathbf{p}_2$ or $\mathbf{p}_3$, and outputs less than zero for all other inputs
- From the problem statement, we know that the network will need to have two inputs and one output
  - For simplicity, we will use only two neurons in the first layer (two basis functions), since this will be sufficient to solve the XOR problem
    - the rows of the first-layer weight matrix will create centers for the two basis functions – we will choose the centers to be equal to the patterns $\mathbf{p}_2$ and $\mathbf{p}_3$

Pattern classification example: the XOR

- By centering a basis function at each pattern, we can produce maximum network outputs there
- The first layer weight matrix is: $\mathbf{W}^1 = \begin{bmatrix} \mathbf{p}_2^T \\ \mathbf{p}_3^T \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$
- The choice of the bias in the first layer depends on the width that we want for each basis function
  - For this problem, we would like the network function to have two distinct peaks at $\mathbf{p}_2$ and $\mathbf{p}_3$. Therefore, we don't want the basis functions to overlap too much. The centers of the basis functions are each a distance of √2 from the origin. We want the basis function to drop significantly from its peak in this distance
    - If we use a bias of 1, we would get the following reduction in that distance: $\alpha = e^{-n\hat{}2} = e^{-(1 * \sqrt{2})\hat{}2} = e^{-2} = 0.1353$
    - Therefore, each basis function will have a peak of 1 at the centers, and will drop to 0.1353 at the origin – this will work for our problem

Pattern classification example: the XOR

- So we select the first layer bias vector to be:

$$\mathbf{b}^1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

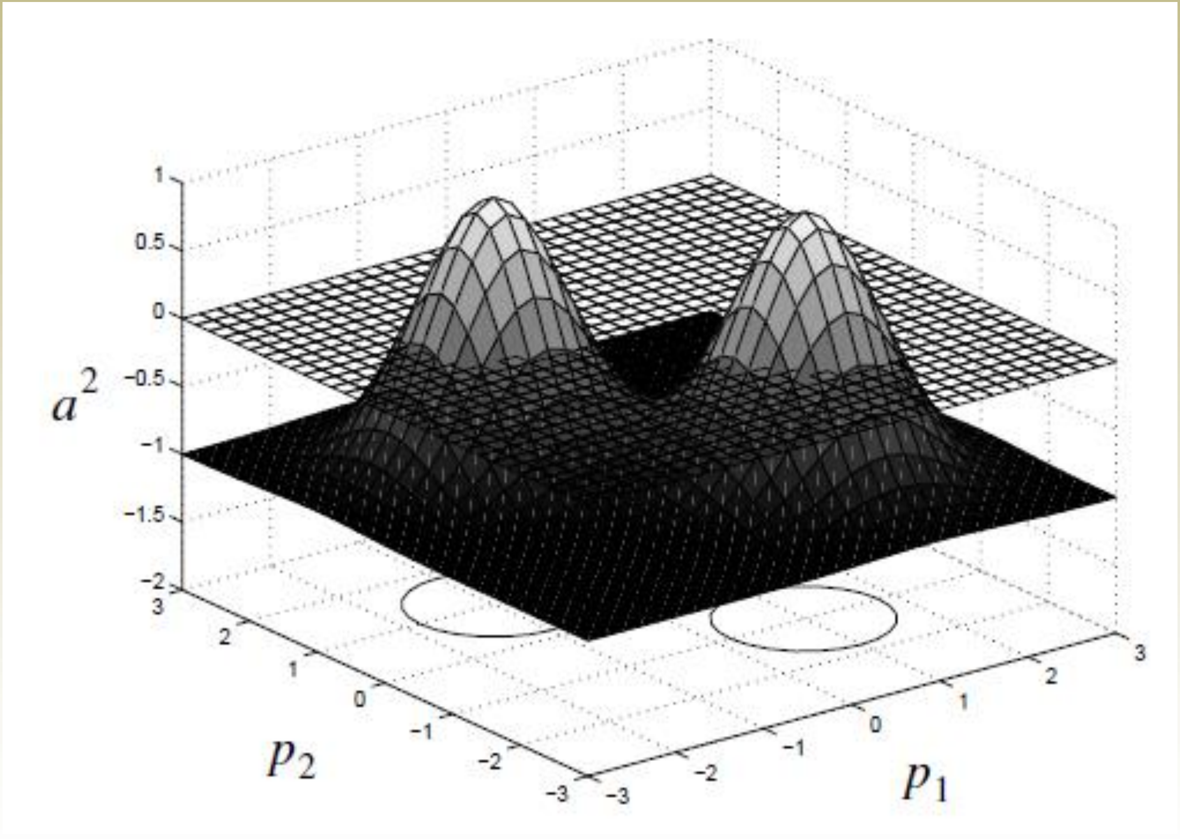
- [The original basis function response ranges from 0 to 1] We want the output to be negative for inputs much different than $\mathbf{p}_2$ and $\mathbf{p}_3$, so we will use a bias of -1 for the second layer, and we will use a value of 2 for the second layer weights, in order to bring the peaks back up to 1
- The second layer weights and biases then become:

$$\mathbf{W}^2 = [2\ 2], \quad b^2 = [-1]$$

Pattern classification example: the XOR

For these network parameter, the response is:

- The figure also shows where the surface intersects the plane at $a^2=0$, which is where the decision takes place
- This is also indicated by the contours shown underneath the surface. These are the function contours where $a^2=0$. They are almost circles that surround the $\mathbf{p}_2$ and $\mathbf{p}_3$ vectors. This means that the network output will be greater than 0 only when the input vector is near the $\mathbf{p}_2$ and $\mathbf{p}_3$ vectors



RBF network training by clustering

- There is another approach for selecting the weights and biases in the first layer of the RBF network: by clustering
  - k-means
  - Competitive layer of Kohonen (subsequent lectures)
  - Self-Organizing Feature Map (subsequent lectures)

- If we take the input vectors from the training set and perform a clustering operation on them
  - the resulting prototypes (cluster centers) could be used as centers for the RBF network
  - we could compute the variance of each individual cluster and use that number to calculate an appropriate bias to be used for the corresponding neuron

RBF network training by clustering

- The clustering procedure (finding the centers) will insure that we will have basis functions located in areas where input vectors are most likely to occur
- In addition to selecting the first layer weights, the clustering process can provide us with a method for determining the first layer biases
  - For each neuron (basis function), locate the input vectors from the training set that are closest to the corresponding weight vector (center)
  - Then compute the average distance between the center and its neighbors

$$dist_i = \frac{1}{n_c}\left(\sum_{j=1}^{n_c} ||\mathbf{p}_j^i - {}_i\mathbf{w}^1||^2\right)^{\frac{1}{2}}$$

where $\mathbf{p}_1^i$ is the input vector that is closest to ${}_i\mathbf{w}^1$, and $\mathbf{p}_2^i$ is the next closest input vector

RBF network training by clustering

- From these distances, we could set the first layer biases as follows:

$$b_i^1 = \frac{1}{\sqrt{2} dist_i}$$

- Therefore, when a cluster is wide, the corresponding basis function will be wide as well
  - Notice that in this case each bias in the first layer will be different. This should provide a network that is more efficient in its use of basis functions than a network with equal biases

- After the weights and biases of the first layer are determined, linear least squares is used to find the second layer weights and biases

RBF network training by clustering

- There is a potential drawback to the clustering method for designing the first layer of the RBF network: The method only takes into account the distribution of the input vectors; it does not consider the targets
  - It is possible that the function we are trying to approximate is more complex in regions for which there are fewer inputs
  - For this case, the clustering method will not distribute the centers appropriately
  - On the other hand, one would hope that the training data is located in regions where the network will be most used, and therefore the function approximation will be most accurate in those areas

RBF network training by nonlinear optimization

- It is also possible to train RBF networks (in the same manner as MLP networks) using nonlinear optimization techniques, in which all weights and biases in the network are adjusted at the same time
  - These methods are not generally used for the full training of RBF networks, because these networks tend to have many more unsatisfactory local minima in their error surfaces
- However, nonlinear optimization can be used for the fine-tuning of the network parameters, after initial training by one of the two-stage methods we have presented

- We will indicate how the basic backpropagation algorithm for computing the gradient in MLP networks can be modified for RBF networks

RBF network training by nonlinear optimization

- The net input for the second layer of the RBF network has the same form as its counterpart in the MLP network,
- but the first layer net input has a different form:

$$n_i^1 = ||\mathbf{p} - {}_i\mathbf{w}^1|| b_i^1 = b_i^1 \sqrt{\sum_{j=1}^{S^1} (p_j - w_{i,j}^1)^2}$$

- If we take the derivative of this function with respect to the weights and biases, we get:

$$\frac{\partial n_i^1}{\partial w_{i,j}^1} = b_i^1 \frac{1/2}{\sqrt{\sum_{j=1}^{S^1} (p_j - w_{i,j}^1)^2}} 2(p_j - w_{i,j}^1)(-1) = \frac{b_i^1 (w_{i,j}^1 - p_j)}{||\mathbf{p} - {}_i\mathbf{w}^1||}$$

$$\frac{\partial n_i^1}{\partial b_i^1} = ||\mathbf{p} - {}_i\mathbf{w}^1||$$

RBF network training by nonlinear optimization

- This produces the modified gradient equations for Layer 1 of the RBF network

$$\frac{\partial \hat{F}}{\partial w^1_{i,j}} = s^1_i \frac{b^1_i (w^1_{i,j} - p_j)}{||\mathbf{p} - {}_i\mathbf{w}^1||}$$

$$\frac{\partial \hat{F}}{\partial b^1_i} = s^1_i ||\mathbf{p} - {}_i\mathbf{w}^1||$$