Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –
Δημήτριος Κατσαρός

@ Τμ. ΗΜΜΥ
Πανεπιστήμιο Θεσσαλίας

Διάλεξη 16η

RBF Network training with Linear Least Squares (LLS)

Radial Basis Network basics recap

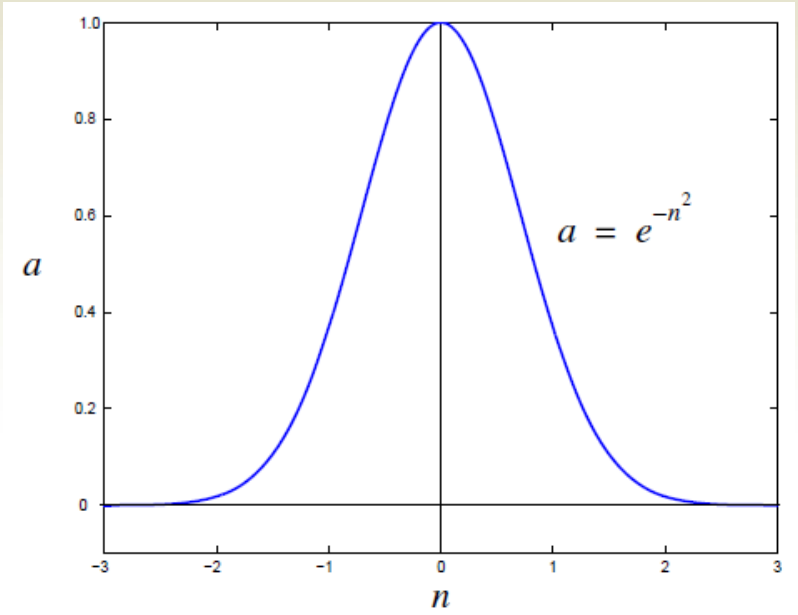
- The radial basis network is a two-layer network. There are two major distinctions between the *radial basis function* (RBF) *network* and a two layer perceptron:
  1. In layer 1 of the RBF network, instead of performing an inner product operation between the weights and the input (matrix multiplication), 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 recap

- 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:

$$a = e^{-n^2}$$

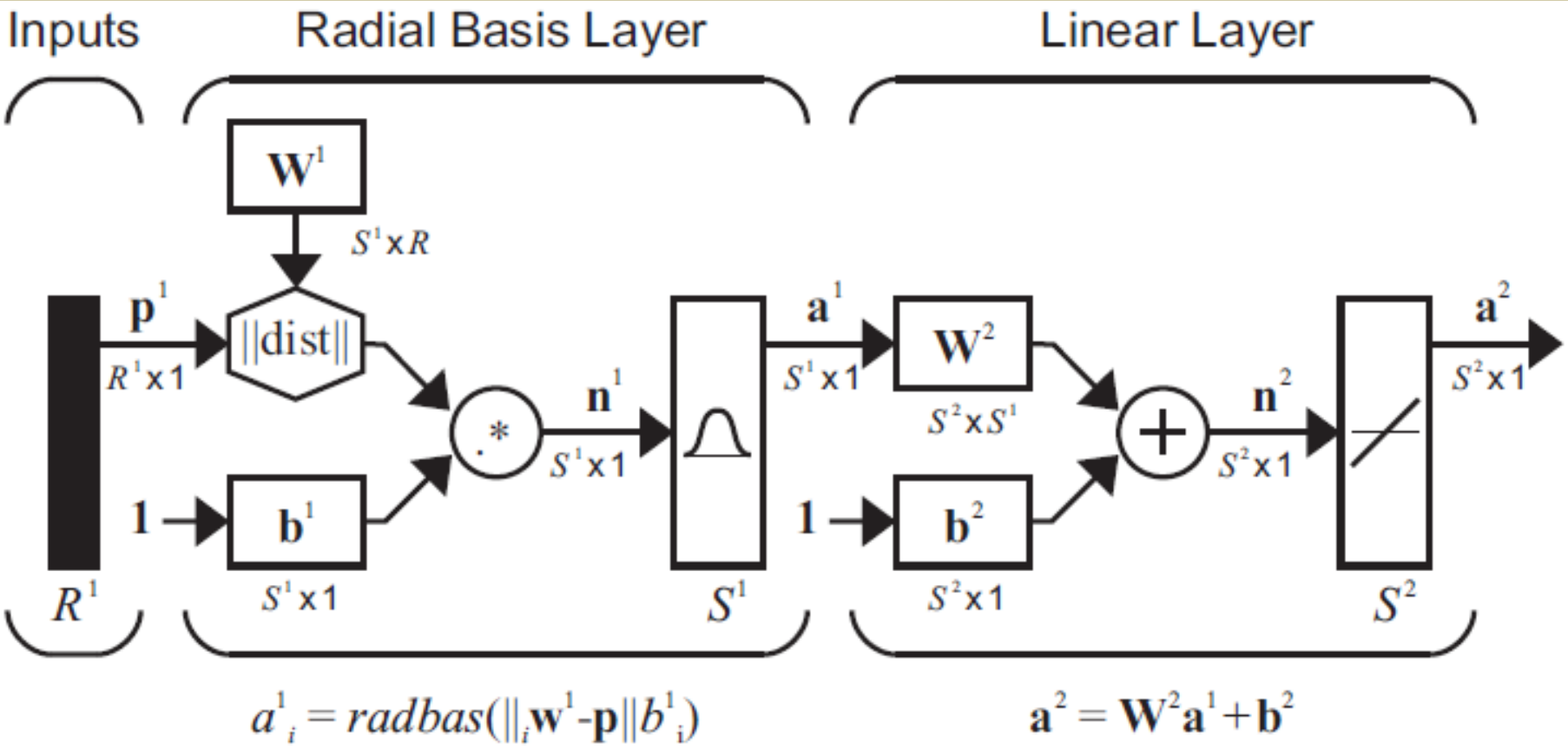
A key property of this function is that it is *local* – the output is close to zero if you move very far in either direction from the center point. This is in contrast to the global sigmoid functions, whose output remains close to 1 as the net input goes to infinity



Radial Basis Network basics recap

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

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



Training RBF networks

- RBF networks can be trained using gradient-based algorithms
  - However, because of the local nature of the transfer function and the way in which the first layer weights and biases operate, there tend to be many more unsatisfactory local minima in the error surfaces of RBF networks than in those of MLP networks
    - For this reason, gradient-based algorithms are often unsatisfactory for the complete training of RBF networks
- The most commonly used RBF training algorithms have two stages, which treat the two layers of the RBF network separately
  - The algorithms differ mainly in how the first layer weights and biases are selected. Once the first layer weights and biases have been selected, the second layer weights can be computed in one step, using a linear least-squares algorithm. We will discuss linear least squares in this lecture

Training RBF networks

- The simplest of the two-stage algorithms arranges the centers (first layer weights) in a grid pattern throughout the input range and then chooses a constant bias so that the basis functions have some degree of overlap
  - This procedure is not optimal, because the most efficient approximation would place more basis functions in regions of the input space where the function to be approximated is most complex.
  - Also, for many practical cases the full range of the input space is not used, and therefore many basis functions could be wasted
  - One of the drawbacks of the RBF network, especially when the centers are selected on a grid, is that they suffer from the *curse of dimensionality*. This means that as the dimension of the input space increases, the number of basis functions required increases geometrically

Training RBF networks

- Another method for selecting the centers is to select some random subset of the input vectors in the training set. This ensures that basis centers will be placed in areas where they will be useful to the network. Due to the randomness of the selection, this procedure is not optimal
- In this lecture we will assume that the first layer weights and biases of the RBF network are fixed
  - This can be done by fixing the centers on a grid, or by randomly selecting the centers from the input vectors in the training data set (or by using a clustering method).
  - When the centers are randomly selected, all of the biases can be computed using the following formula (d_{max} is the maximum distance between neighboring centers):

$$b_i^1 = \frac{\sqrt{S^1}}{d_{max}}$$

Training RBF networks with LLS

- Once the first layer parameters have been set, the training of the second layer weights and biases is equivalent to training a linear network, as we did for ADALINE
- For example, consider that we have the following training points: $\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$
- The net input and output of the first layer are as follows:

$$n_{i,q}^1 = \left\| \mathbf{p}_q - {}_i\mathbf{w}^1 \right\| b_i^1$$

$$\mathbf{a}_q^1 = \mathbf{radbas}(\mathbf{n}_q^1)$$

- Since the first layer weights and biases will not be adjusted, the training data set for the second layer then becomes: $\{\mathbf{a}_1^1, \mathbf{t}_1\}, \{\mathbf{a}_2^1, \mathbf{t}_2\}, \ldots, \{\mathbf{a}_Q^1, \mathbf{t}_Q\}$

Training RBF networks with LLS

- The second layer response is linear:

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

- We want to select the weights and biases in this layer to minimize the sum square error performance index over the training set:

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q^2)^T (\mathbf{t}_q - \mathbf{a}_q^2)$$

- Our derivation of the solution to this linear least squares problem will follow the linear network derivation we developed for ADALINE

Training RBF networks with LLS

- To simplify the discussion, we will assume a scalar target, and we will lump all of the parameters we are adjusting, including the bias, into one vector:

$$\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w}^2 \\ b^2 \end{bmatrix}$$

- Similarly, we include the bias input "1" as a component of the input vector:

$$\mathbf{z}_q = \begin{bmatrix} \mathbf{a}_q^1 \\ 1 \end{bmatrix}$$

- Now the network output, which we usually write in the form:

$$a_q^2 = ({}_1\mathbf{w}^2)^T \mathbf{a}_q^1 + b^2$$

can be written as:

$$a_q = \mathbf{x}^T \mathbf{z}_q$$

Training RBF networks with LLS

- This allows us to conveniently write out an expression for the sum square error:

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (e_q)^2 = \sum_{q=1}^{Q} (t_q - a_q)^2 = \sum_{q=1}^{Q} (t_q - \mathbf{x}^T \mathbf{z}_q)^2$$

- To express this in matrix form, we define the following matrices:

$$\mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_Q \end{bmatrix}, \mathbf{U} = \begin{bmatrix} {}_1\mathbf{u}^T \\ {}_2\mathbf{u}^T \\ \vdots \\ {}_Q\mathbf{u}^T \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1^T \\ \mathbf{z}_2^T \\ \vdots \\ \mathbf{z}_Q^T \end{bmatrix}, \mathbf{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_Q \end{bmatrix}$$

- The error can now be written: $\mathbf{e} = \mathbf{t} - \mathbf{U}\mathbf{x}$

  and the performance index becomes: $F(\mathbf{x}) = (\mathbf{t} - \mathbf{U}\mathbf{x})^T (\mathbf{t} - \mathbf{U}\mathbf{x})$

Training RBF networks with LLS

- To avoid overfitting, we modify this equation into the following ($\rho=\alpha/\beta$):

$$F(\mathbf{x}) = (\mathbf{t}-\mathbf{U}\mathbf{x})^T(\mathbf{t}-\mathbf{U}\mathbf{x}) + \rho\sum_{i=1}^{n} x_i^2 = (\mathbf{t}-\mathbf{U}\mathbf{x})^T(\mathbf{t}-\mathbf{U}\mathbf{x}) + \rho\mathbf{x}^T\mathbf{x}$$

- By expanding the above, we get:

$$F(\mathbf{x}) = (\mathbf{t}-\mathbf{U}\mathbf{x})^T(\mathbf{t}-\mathbf{U}\mathbf{x}) + \rho\mathbf{x}^T\mathbf{x} = \mathbf{t}^T\mathbf{t} - 2\mathbf{t}^T\mathbf{U}\mathbf{x} + \mathbf{x}^T\mathbf{U}^T\mathbf{U}\mathbf{x} + \rho\mathbf{x}^T\mathbf{x}$$
$$= \mathbf{t}^T\mathbf{t} - 2\mathbf{t}^T\mathbf{U}\mathbf{x} + \mathbf{x}^T[\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]\mathbf{x}$$

- This is of a quadratic form

$$F(\mathbf{x}) = c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$$

$$c = \mathbf{t}^T\mathbf{t},\ \mathbf{d} = -2\mathbf{U}^T\mathbf{t} \text{ and } \mathbf{A} = 2[\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]$$

Training RBF networks with LLS

- The characteristics of the quadratic function depend primarily on the Hessian matrix (matrix **A** in our case)
  - E.g., if the eigenvalues of the Hessian are all positive, then the function will have one unique global minimum
- It can be shown that this matrix is either positive definite or positive semidefinite,
  - I.e., it can never have negative eigenvalues
- We are left with two possibilities
  - If the Hessian matrix has only positive eigenvalues
    - the performance index will have one unique global minimum
  - If the Hessian matrix has some zero eigenvalues
    - the performance index will either have a weak minimum or no minimum, depending on the vector **d**
    - In this case, it must have a minimum, since is a sum square function, which cannot be negative

Training RBF networks with LLS

- Lets locate the stationary point
- We calculate the following:

$$\nabla F(\mathbf{x}) = \nabla\left(c + \mathbf{d}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}\right) = \mathbf{d} + \mathbf{A}\mathbf{x} = -2\mathbf{U}^T\mathbf{t} + 2[\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]\mathbf{x}$$

- The stationary point of can be found by setting the gradient equal to zero:

$$-2\mathbf{Z}^T\mathbf{t} + 2[\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]\mathbf{x} = 0 \quad \Rightarrow \quad [\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]\mathbf{x} = \mathbf{U}^T\mathbf{t}$$

- Therefore, the optimum weights **x*** can be computed from:

$$[\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]\mathbf{x}^* = \mathbf{U}^T\mathbf{t}$$

- If the Hessian matrix is positive definite, there will be a unique stationary point, which will be a strong minimum:

$$\mathbf{x}^* = [\mathbf{U}^T\mathbf{U} + \rho\mathbf{I}]^{-1}\mathbf{U}^T\mathbf{t}$$