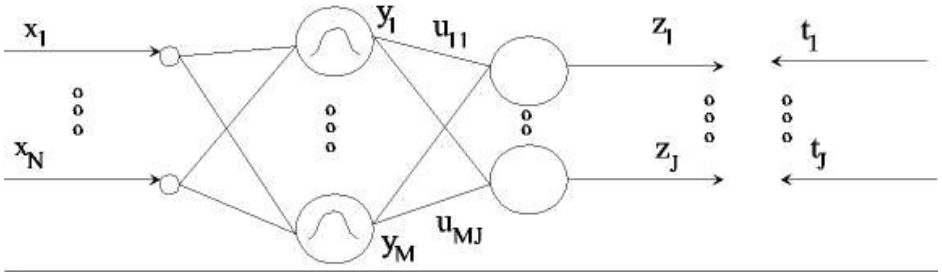
Radial Basis Function Neural Network Tutorial

The Architecture of RBFNN's

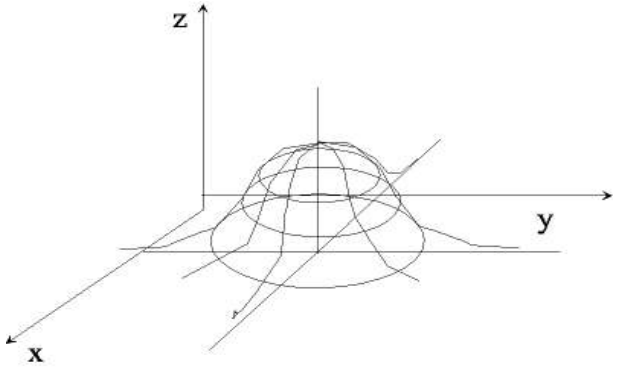
The figure below shows a radial basis function neural network. The bell shaped curves in the hidden nodes indicate that each hidden layer node represents a bell shaped *radial basis function* that is centered on a vector in the feature space. There are no weights on the lines from the input nodes to the hidden nodes. The input vector is fed to each m-th hidden node where it is put through the that nodes radial basis function

$$y_m = f_m(\mathbf{x}) = \exp[-\|\mathbf{x} - \mathbf{c}_m\|^2/(2\sigma^2)] \quad (1)$$

where $\|\mathbf{x} - \mathbf{c}_m\|^2$ is the square of the distance between the input feature vector $\mathbf{x}$ and the center vector $\mathbf{c}_m$ for that radial basis function.



The values $\{y_m\}$ are the outputs from the radial basis functions. These radial basis functions on a 2-dimensional feature space have the form shown in the simple graph below. The values equidistant from the center in all directions have the same values, so this is why these are called radial basis functions.



The outputs from the hidden layer nodes are weighted by the weights on the lines and the weighted sum is computed at each j-th output node as

$$z_j = (1/M)\Sigma_{(m = 1,M)}\ u_{mj}y_m \quad (2)$$

Training the RBFNN

The mean square error function that is to be minimized by adjusting the parameters $\{u_{mj}\}$ is similar to the one for BPNN except that this one is much simpler to minimize. There is only one set of parameters instead of two as was the case for BPNNs. Upon suppressing the index q we have

$$E = (1/J)\,\Sigma_{(j=1,J)}\,(t_j - z_j)^2 \;=\; (1/J)\,\Sigma_{(j=1,J)}\,(t_j - (1/M)\Sigma_{(m=1,M)}\,u_{mj}y_m)^2 \tag{3}$$

Thus

$$\partial E/\partial u_{mj} = (\partial E/\partial z_j)(\partial z_j/\partial u_{mj}) = [(-2/J)\,\Sigma_{(j=1,J)}\,(t_j - z_j)](y_m/M) \tag{4}$$

Upon putting this into the steepest descent method

$$u_{mj}^{(k+1)} \;=\; u_{mj}^{(k)} + [2\eta/(JM)]\,\Sigma_{(j=1,J)}\,(t_j - z_j)]\,y_m \tag{5}$$

where η is the learning rate, or step size, as before. Upon training over all Q feature vector inputs and their corresponding target output vectors, Equation (5) becomes

$$u_{mj}^{(k+1)} \;=\; u_{mj}^{(k)} + [2\eta/(JM)]\Sigma_{(q=1,Q)}\;\Sigma_{(j=1,J)}\,(t_j^{(q)} - z_j^{(q)})]\,y_m^{(q)} \tag{6}$$

There is still some missing information that we must have before we can implement an algorithm for training a RBFNN on a given data set $\{\{\mathbf{x}^{(q)}: q = 1,...,Q\}, \{\mathbf{t}^{(q)}: q = 1,...,Q\}\}$ (here the feature vectors for training (the exemplar vectors) and paired with the target vectors by the index q). We still don't know the center vectors $\{\mathbf{c}^{(m)}: m = 1,...,M\}$ on which to center the radial basis functions. We also don't know M and we don't know the spread parameter σ.

There are different methods to get these. The original method is to use the exemplar vectors $\{\mathbf{x}^{(q)}: q = 1,...,Q\}$ as the centers by putting $\mathbf{c}^{(m)} = \mathbf{x}^{(q)}$ for m = 1,...,Q. This is satisfactory when the exemplar feature vectors are scattered well over the feature space, which means they must be numerous and cover all possible classes.

Another method is to use the exemplar vectors as the first Q centers, but then to generate many more centers at random in the feature space. Then the distances between centers are computed and any center that is too close to another center is eliminated. However, we should have significantly more centers than there are classes. The threshold for eliminating can be, say, 0.4 times the average distance between centers. Then σ can be taken to be from 0.25 to 0.5 times the average distance of the remaining centers (the radial basis functions will overlap some, but not too much).

An Algorithm for the RBFNN

To implement the indicated algorithm of Equation (6) we first read in the file of data that includes values for N, M, J, Q, the feature vectors and their target vectors, draw the parametric

weights $\{u_{mj}\}$ at random between -0.5 and 0.5, and iterate on the parameters as described above.

Step 1: Read the data file to get N, J, Q, the feature vectors and their target vector, input the number of iterations I, set i = 0, set Q centers of RBF's as the Q exemplar vectors, put M = 2Q.

Step 2. Find average distance between centers, eliminate centers too close to another center, set M as final set of centers, compute σ, and draw the parameters $\{u_{mj}\}$ at random between -0.5 and 0.5

Step 3. Compute $\{y_m\}$ and $\{z_j\}$, and then compute E

Step 4. Update all parameters u_{mj} for all m and j at the current iteration by Equation (6)

Step 5. Compute $\{y_m\}$ and $\{z_j\}$, and then compute the new value for E

Step 6. If new E is less than the old E then increase η else decrease it

Step 7. Increment iteration i, if i < I then go to Step 4, else stop