SNN Example 14 - Radial Basis Function Networks

STATISTICA Neural Networks supports a number of network architectures, of which Multilayer Perceptrons (MLPs) are perhaps the best known.

The **second most commonly used neural network architecture** is the Radial Basis Function (RBF) network (see Haykin, 1994; Bishop, 1995).

The units in a multilayer perceptron each perform a linear transformation on the input vector (set of values entering the unit); specifically, they perform a weighted sum of the inputs (i.e. they form the dot product of the input vector with the weight vector), and subtract from this sum the threshold. In *STATISTICA Neural Networks*, this is referred to as the dot product Synaptic (Post Synaptic Potential) function. The result is then passed through the non-linear activation function.

The Dot Product synaptic function means that multilayer perceptrons essentially work by dividing up pattern space using hyperplanes (in two-dimensional space, a hyperplane is just a line) - a linear operation. Nonlinearity is added by the typically sigmoidal activation function.

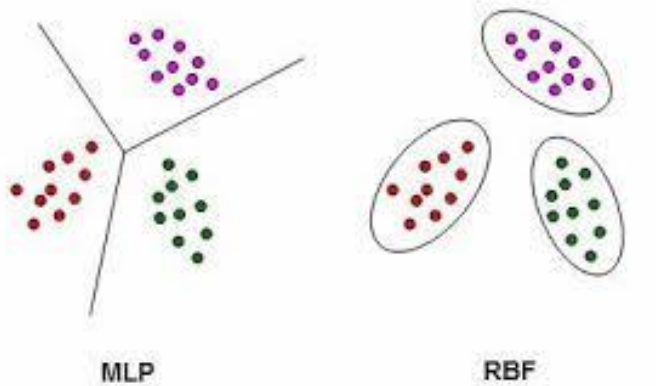
In contrast to the multilayer perceptron's linear approach, a radial basis function network uses a Radial Synaptic function. Each unit measures the square of the distance of the input vector from the weight vector. This distance is then multiplied by the threshold (actually, therefore, a deviation) before being passed through the activation function. Thus, radial basis function networks work by dividing up pattern space using hyperspheres (in two-dimensional space, a hypersphere is just a circle).

The two approaches have contrasting advantages and disadvantages. The radial approach is very localized, whereas the linear approach is active over the entire pattern space.

Consequently, **RBF networks tend to need more units than MLPs**, but MLPs may make unjustified extrapolations if data that is unlike any of the training data is used, whereas an RBF will always have a near-zero (or near sample mean) response in this case.

Theory also indicates that an MLP may need two hidden layers to solve some problems, and even more hidden layers may sometimes be efficient. In contrast, an RBF with one hidden layer is always adequate.

Comparison of RBF Networks and MLPs



- **Both are examples of nonlinear layered feedforward networks**
- **Both are universal approximators**
- **There always exists an RBF network that can mimic a given MLP, or vice versa. But, these networks are different in several important respects:**
  1. An RBF network has a single hidden layer, whereas an MLP may have one or more
  2. Typically, computation nodes of an MLP share a common neural model. But in an RBF network they are different and serve different purposes
  3. In an RBF network, hidden layer is nonlinear and output layer is linear. But in an MLP used for classification, usually all units are nonlinear. When the MLP is used for solving nonlinear regression problems, a linear output layer is preferred.
  4. Argument of the activation functions in an RBF network computes distance between input and a center. But activation functions in an MLP compute an inner product
  5. MLPs construct global approximations but RBF networks (with certain specific choices) construct local approximations

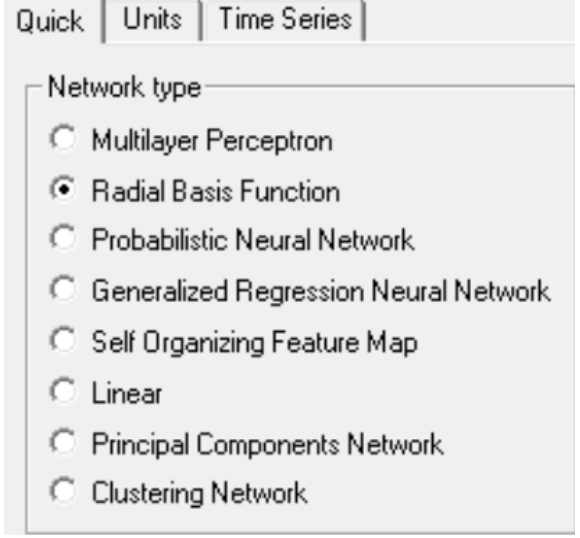
An RBF always has **exactly three layers**: the input layer, the hidden layer that contains radial units, and (in the standard formulation) a linear output layer. In *STATISTICA Neural Networks*, the linear output layer is represented by having a Dot Product synaptic function, and the Identity activation function. The nonlinearity of the radial units allows this output layer to be linear without losing the ability to handle nonlinear functions. *STATISTICA Neural Networks* includes standard linear optimization techniques that allow the linear output layer to be optimized, provided that the earlier layers of the network are first fixed.

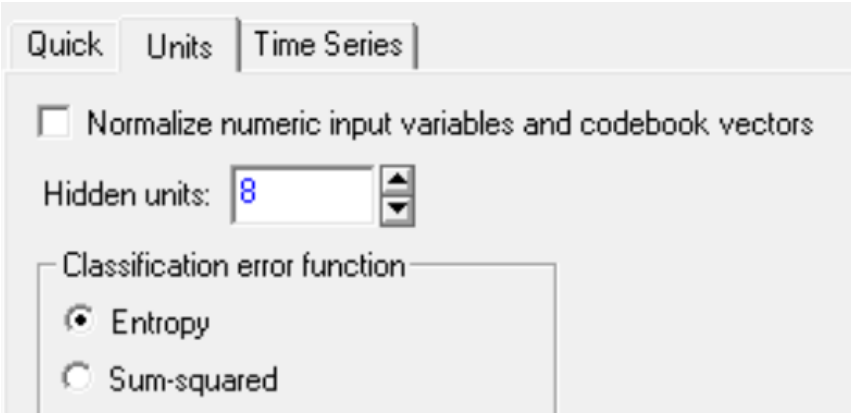
The approach to RBF training is therefore quite different to that used in an MLP.
First, the **radial centers** and their deviations (spreads) are set using unsupervised techniques (i.e., techniques that consider only the input variables in the training data). Essentially, the idea is to pick centers that lie at the heart of clusters of training data, with deviations selected to reflect the density of the data.
Second, the linear output layer is optimized using the pseudo-inverse technique.

Although well-suited to **regression problems**, a standard RBF with a linear output layer is less suited to classification. The output neuron activation levels are not constrained to sum to 1.0, and so cannot be interpreted as probabilities. However, as with MLPs, by modifying the networks to use an entropy-based error function, and an appropriate output layer activation function (either softmax, or logistic, depending on the number of output neurons), the network can be interpreted as estimating probabilities, on the assumption that the underlying distribution is from the exponential family.

Using an RBF Network for the Iris Problem. To create an RBF network, use the Custom Network Designer as usual, but change the Network type to Radial Basis Function. As with Multilayer Perceptrons, you can specify the number of hidden units on the Units tab (you cannot change the number of hidden layers in an RBF, however).

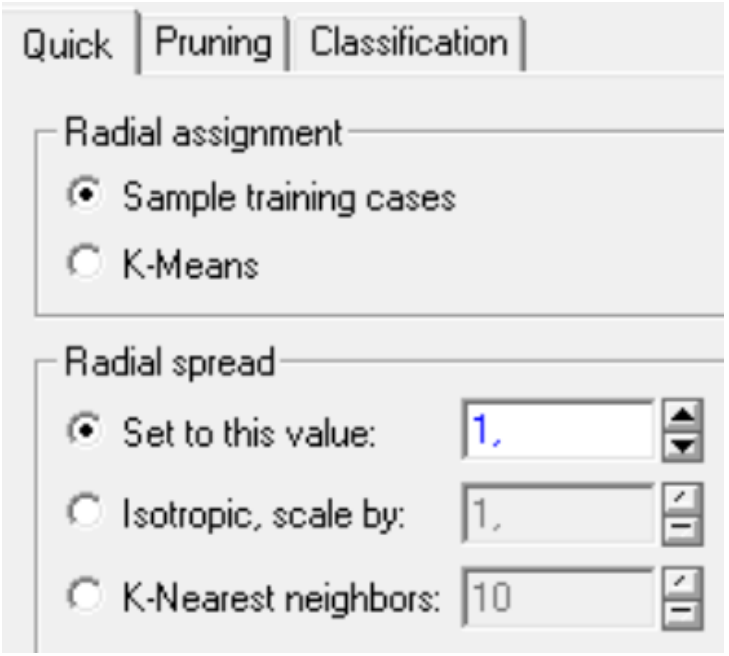
The number of hidden units is typically much larger in an RBF than in an MLP. However, in this case quite a small number of hidden units (ten, or even less) is adequate.



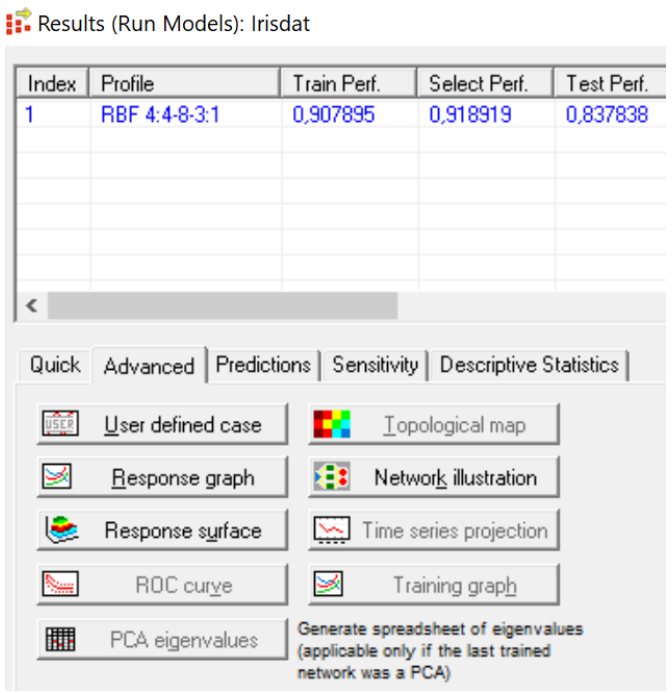


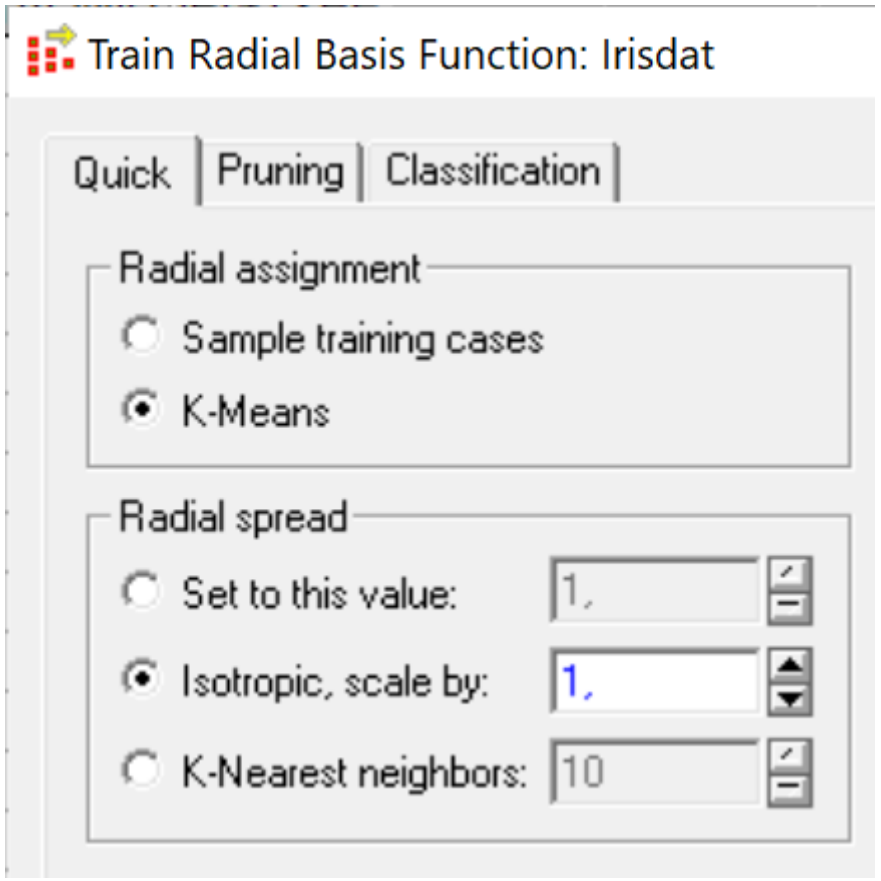
STATISTICA Neural Networks supports both standard sum-squared and entropy based RBF networks.

One disadvantage of entropy-based RBFs is that linear optimization can no longer be deployed on the output layer. Instead, STATISTICA Neural Networks uses **Conjugate Gradient Descent**. However, although slower than linear optimization, the algorithm does converge in a reasonable amount of time and, unlike the case of Multilayer Perceptron networks, the error function is unimodal, so that convergence is guaranteed.



Training an RBF Network. As you will see, another advantage of radial basis function networks is that they **train quite rapidly**, and in this case you will probably find the performance comparable to that of the Multilayer Perceptron.
A disadvantage is that the pseudo-inverse algorithm is prone to error if the radial deviations are too small, so that the Isotropic and K-Means deviation algorithms must be used with care. If you do run into numerical difficulties, use the custom training facility to optimize the output layer using an iterative algorithm such as conjugate gradient descent.





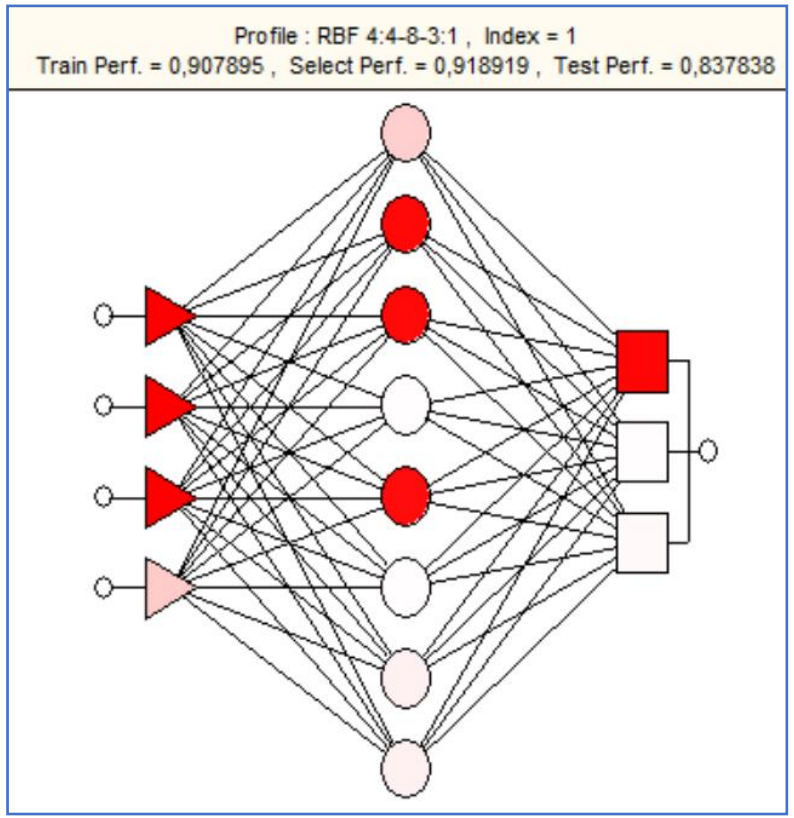
Select the K-Means option button in the Radial assignment group box; this assigns cluster centers to the radial units;

Select the Isotropic, scale by option button in the Radial spread group box; this assigns appropriate deviations based on the number of units and the spread of the training data.

Index	Model Summary Report (Irisdat)											
	Profile	Train Perf.	Select Perf.	Test Perf.	Train Error	Select Error	Test Error	Training/Members	Note	Inputs	Hidden(1)	Hidden(2)
1	RBF 4:4-8-3:1	0,907895	0,918919	0,837838	0,358616	0,340021	0,415015	KM,IS,CG10s		4	8	0

	Classification (1) (Irisdat)		
	IRISTYPE.SETOSA.1	IRISTYPE.VIRGINIC.1	IRISTYPE.VERSICOL.1
Total	50,0000	50,00000	50,00000
Correct	50,0000	39,00000	45,00000
Wrong	0,0000	11,00000	5,00000
Unknown	0,0000	0,00000	0,00000
Correct(%)	100,0000	78,00000	90,00000
Wrong(%)	0,0000	22,00000	10,00000
Unknown(%)	0,0000	0,00000	0,00000

	Confusion Matrix - IRISTYPE(1) (Irisdat)		
	SETOSA	VIRGINIC	VERSICOL
SETOSA.1	50,00000	0,00000	0,00000
VIRGINIC.1	0,00000	39,00000	5,00000
VERSICOL.1	0,00000	11,00000	45,00000



	Sensitivity Analysis - 1 (Irisdat)			
	SEPALLEN	SEPALWID	PETALLEN	PETALWID
Ratio.1	0,985464	0,977271	2,924375	1,139434
Rank.1	3,000000	4,000000	1,000000	2,000000