Radial Basis Function Networks: Applications

Neural Computation : Lecture 15

1. Regularization Theory for RBF Networks
2. RBF Networks for Classification
3. The XOR Problem in RBF Form
4. Interpretation of Gaussian Hidden Units
5. Comparison of RBF Networks with MLPs
6. Real World Application – EEG Analysis

Regularization Theory for RBF Networks

Instead of restricting the number of hidden units, an alternative approach for preventing over-fitting in RBF networks comes from the theory of ***regularization***, which was seen previously to be a method of controlling the smoothness of mapping functions.

One can have a basis function centred on each training data point as in the case of exact interpolation, but add an extra term to the error/cost function which penalizes mappings that are not smooth. For network outputs $y_k(\mathbf{x}^p)$ and sum squared error function, some appropriate regularization function Ω can be introduced to give

$$E = E_{sse} + \lambda\Omega = \tfrac{1}{2}\sum_p \sum_k (t_k^p - y_k(\mathbf{x}^p))^2 + \lambda\Omega$$

where λ is the regularization parameter which determines the relative importance of smoothness compared with error. There are many possible forms for Ω, but the general idea is that mapping functions $y_k(\mathbf{x})$ which have large curvature should have large values of Ω and hence contribute a large penalty in the total error function.

Computing the Regularized Weights

Provided the regularization term is quadratic in the output weights w_{kj}, they can still be found by solving a set of linear equations. For example, the two popular regularizers

$$\Omega = \tfrac{1}{2}\sum_{k,j}(w_{kj})^2 \qquad \text{and} \qquad \Omega = \sum_p \sum_{k,i} \frac{1}{2}\left(\frac{\partial^2 y_k(\mathbf{x}^p)}{\partial x_i^{\,2}}\right)^2$$

both directly penalize large output curvature, and minimizing the error function E leads to solutions for the output weights that are no harder to compute than we had before:

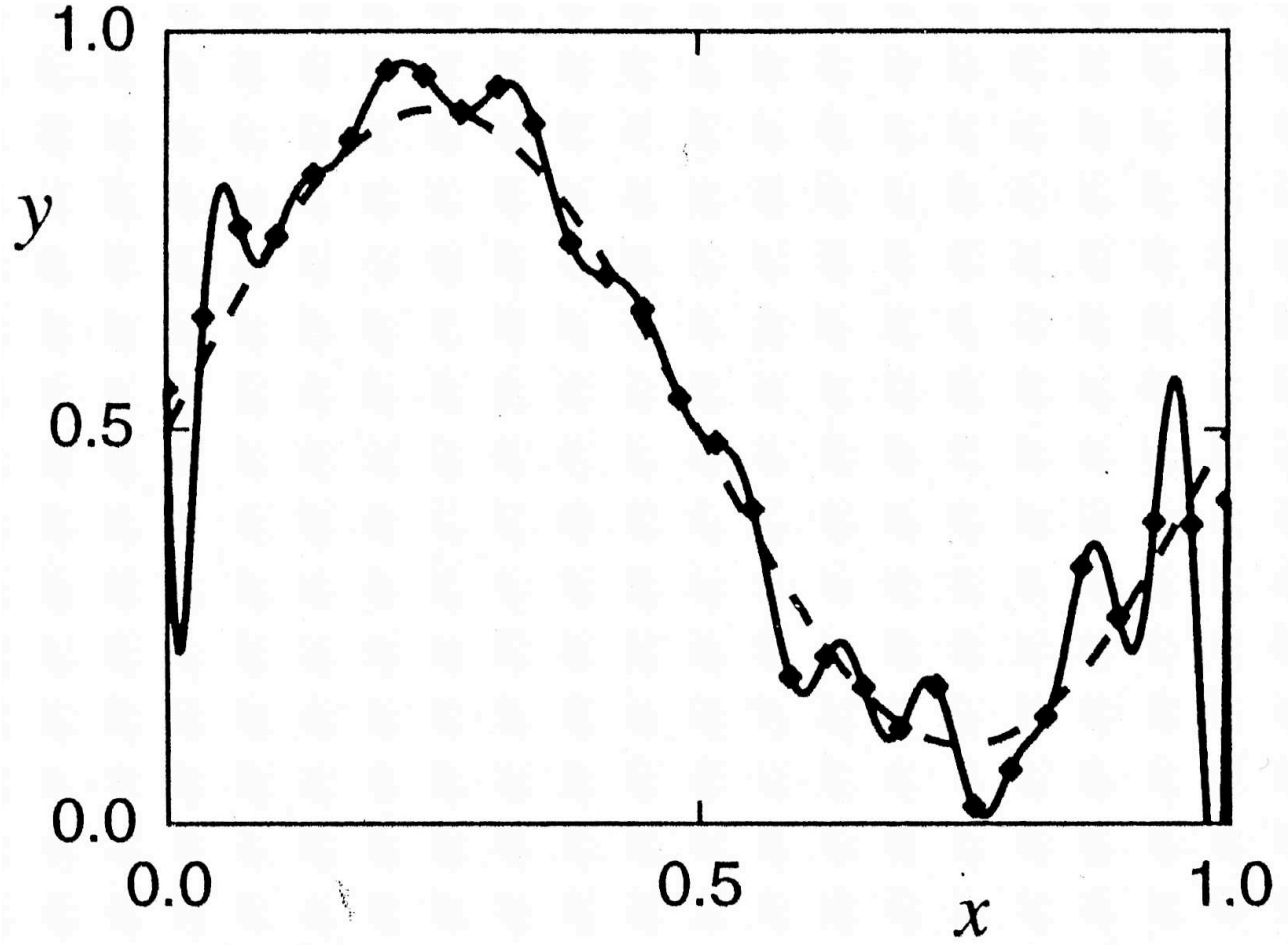
$$\mathbf{W}^{\mathsf{T}} = \mathbf{M}^{-1}\mathbf{\Phi}^{\mathsf{T}}\mathbf{T}$$

We have the same matrices with components $(\mathbf{W})_{kj} = w_{kj}$, $(\mathbf{\Phi})_{pj} = \phi_j(\mathbf{x}^p)$ and $(\mathbf{T})_{pk} = \{t_k^p\}$ as before, but now have different regularized versions of $\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi}$ for the two regularizers:

$$\mathbf{M} = \mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi} + \lambda\mathbf{I} \qquad \text{and} \qquad \mathbf{M} = \mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi} + \lambda\sum_i \frac{\partial^2\mathbf{\Phi}^{\mathsf{T}}}{\partial x_i^{\,2}}\frac{\partial^2\mathbf{\Phi}}{\partial x_i^{\,2}}$$

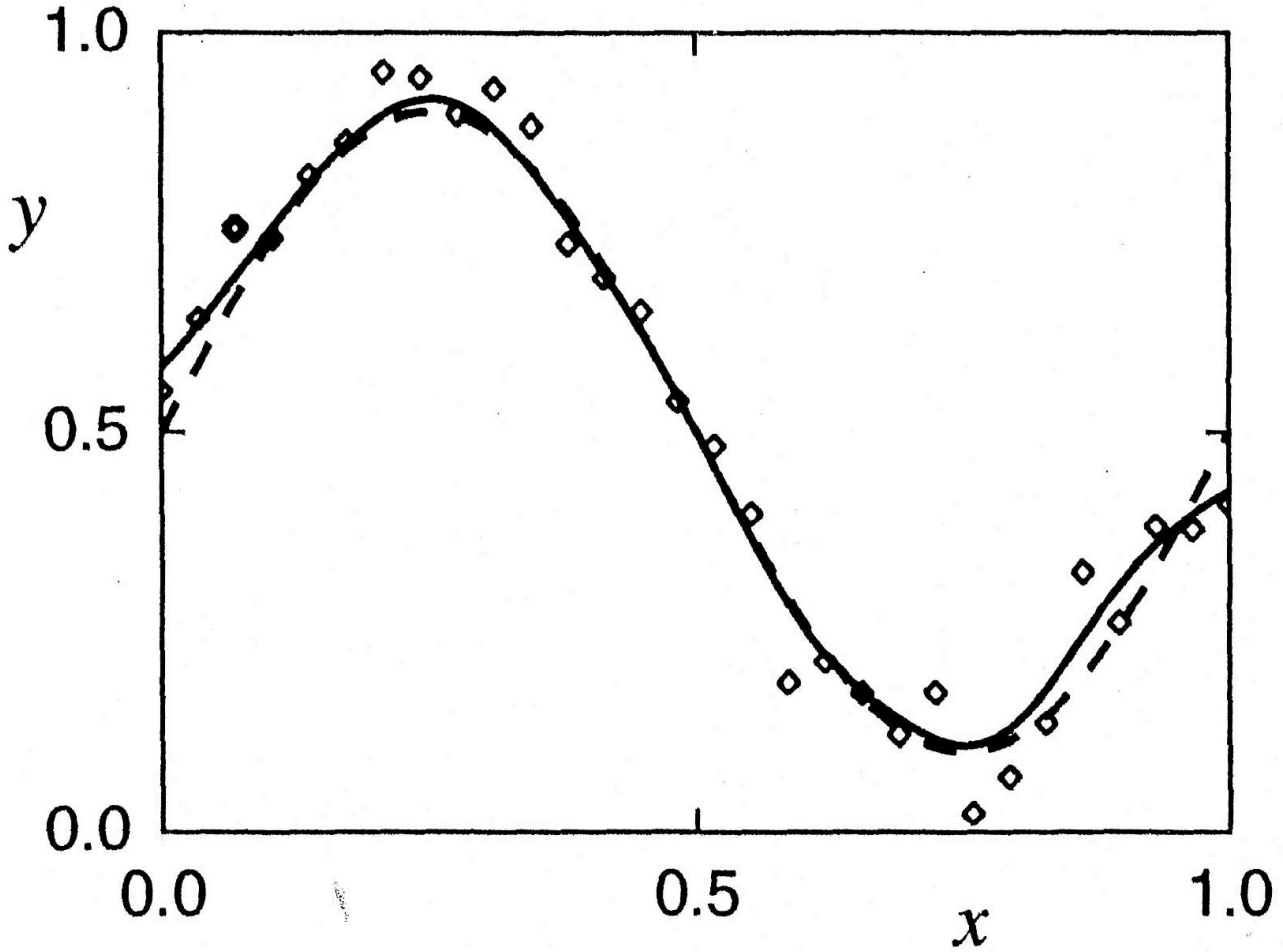
Clearly, for $\lambda = 0$ both reduce to the un-regularized result we derived in the last lecture.

Example 1 : $M = N$, $\sigma = 2d_{ave}$, $\lambda = 0$



From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

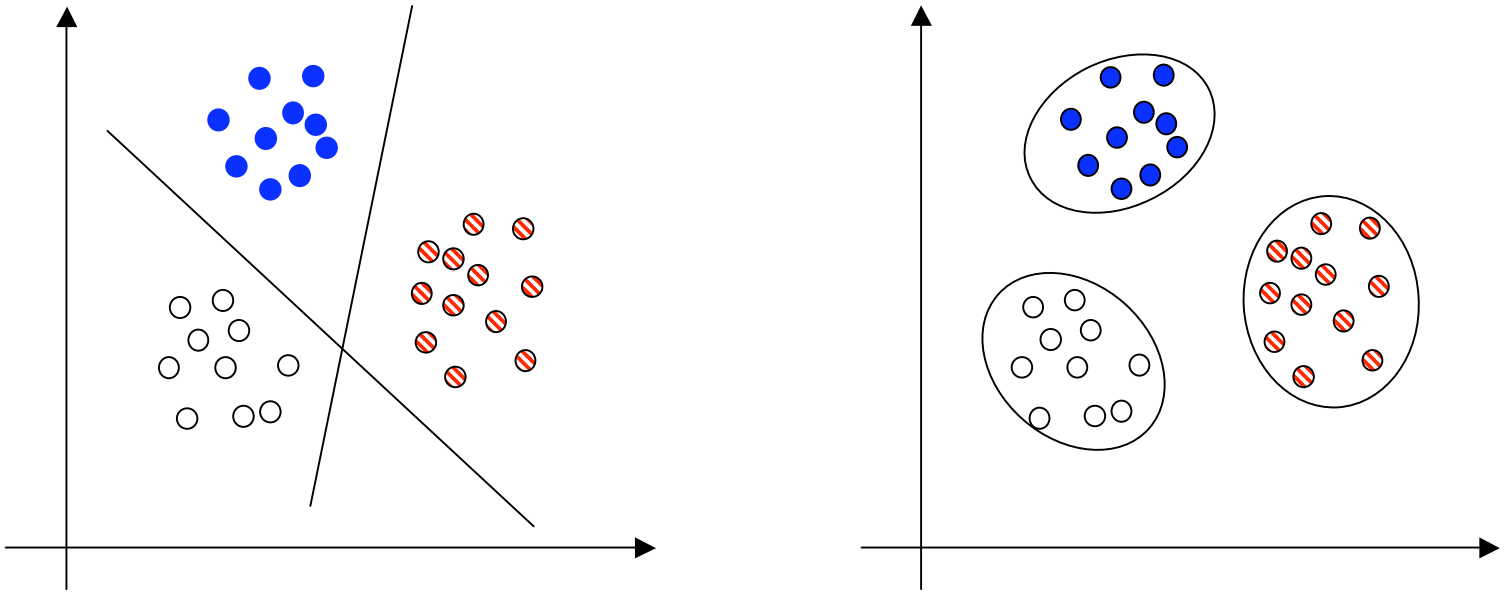
Example 2 : $M = N$, $\sigma = 2d_{ave}$, $\lambda = 40$



From: Neural Networks for Pattern Recognition, C. M. Bishop, Oxford University Press, 1995.

RBF Networks for Classification

So far, the RBF networks have been used for function approximation, but they are also useful for classification problems. Consider a data set that falls into three classes:



An MLP would naturally separate the classes with hyper-planes in the input space (as on the left). An alternative approach would be to model the separate class distributions by localised radial basis functions (as on the right).

Implementing RBF Classification Networks

In principle, it is easy to set up an RBF network to perform classification – one can simply have an output function $y_k(\mathbf{x})$ for each class k with appropriate targets

$$t_k^p = \begin{cases} 1 & \text{if pattern } p \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases}$$

and, when the network is trained, it will automatically classify new patterns.

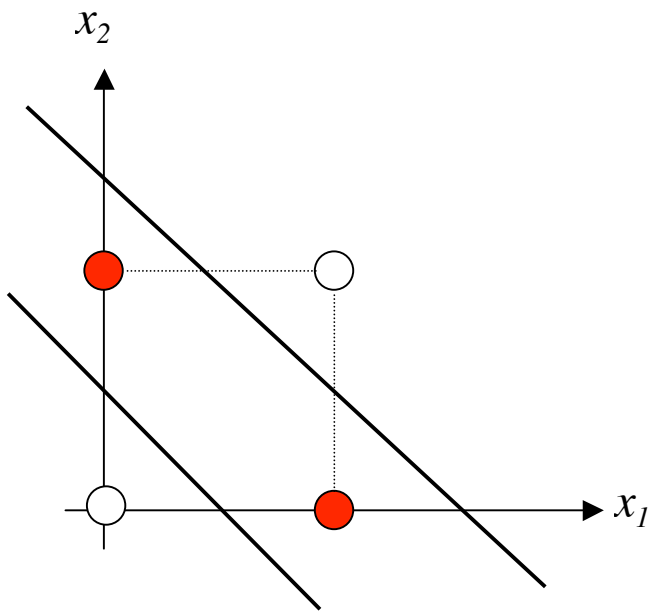
The underlying justification is found in ***Cover's theorem*** which states that "A complex pattern classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low dimensional space". We know that once we have linear separable patterns, the classification problem is easy to solve.

In addition to the RBF network outputting good classifications, it can be shown that the outputs of such a regularized RBF network classifier can also provide good estimates of the ***posterior class probabilities***.

The XOR Problem Revisited

The familiar case of the non-linearly separable XOR function provides a good example:

p	x_1	x_2	t
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0



It was seen before that Single Layer Perceptrons with step or sigmoidal activation functions cannot generate the right outputs, because they can only form a single linear decision boundary. To deal with this problem using Perceptrons, one must either change the activation function, or introduce a non-linear hidden layer to give an Multi Layer Perceptron (MLP).

The XOR Problem in RBF Form

Recall that sensible RBFs are M Gaussians $\phi_j(\mathbf{x})$ centred at random training data points:

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{M}{d_{\max}^2}\|\mathbf{x}-\boldsymbol{\mu}_j\|^2\right) \qquad \text{where} \quad \{\boldsymbol{\mu}_j\} \subset \{\mathbf{x}^p\}$$

To perform the XOR classification in an RBF network, one must begin by deciding how many basis functions are needed. Given there are four training patterns and two classes, $M = 2$ seems a reasonable first guess. Then the basis function centres need to be chosen. The two separated zero targets seem a good random choice, so $\boldsymbol{\mu}_1 = (0,0)$ and $\boldsymbol{\mu}_2 = (1,1)$ and the distance between them is $d_{max} = \sqrt{2}$. That gives the basis functions:

$$\phi_1(\mathbf{x}) = \exp\left(-\|\mathbf{x}-\boldsymbol{\mu}_1\|^2\right) \qquad \text{with} \quad \boldsymbol{\mu}_1 = (0,0)$$

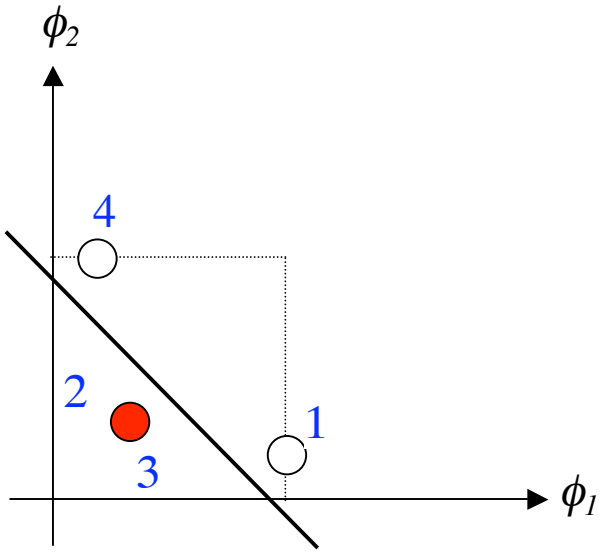
$$\phi_2(\mathbf{x}) = \exp\left(-\|\mathbf{x}-\boldsymbol{\mu}_2\|^2\right) \qquad \text{with} \quad \boldsymbol{\mu}_2 = (1,1)$$

This is hopefully sufficient to transform the problem into a linearly separable form.

The XOR Problem Basis Functions

Since the hidden unit activation space is only two dimensional, it is easy to plot the activations to see how the four input patterns have been transformed:

p	x_1	x_2	ϕ_1	ϕ_2
1	0	0	1.0000	0.1353
2	0	1	0.3678	0.3678
3	1	0	0.3678	0.3678
4	1	1	0.1353	1.0000



It is clear that the patterns are now linearly separable. Note that, in this case, there is no need to increase the dimensionality from the input space to the hidden unit/basis function space – the non-linearity of the mapping is sufficient. **Exercise**: check what happens if you chose a different pair of basis function centres, or one or three centres.

The XOR Problem Output Weights

In this case, there is just one output $y(\mathbf{x})$, with one weight w_j from each hidden unit j, and one bias $-\theta$. So, the network's input-output relation for each input pattern $\mathbf{x}$ is

$$y(\mathbf{x}) = w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) - \theta$$

Thus, to make the outputs $y(\mathbf{x}^p)$ equal the targets t^p, there are four equations to satisfy:

$$\begin{aligned}
1.0000w_1 + 0.1353w_2 - 1.0000\theta &= 0 \\
0.3678w_1 + 0.3678w_2 - 1.0000\theta &= 1 \\
0.3678w_1 + 0.3678w_2 - 1.0000\theta &= 1 \\
0.1353w_1 + 1.0000w_2 - 1.0000\theta &= 0
\end{aligned}$$

Three are different, and there are three variables, so they are easily solved to give

$$w_1 = w_2 = -2.5018 \quad , \quad \theta = -2.8404$$

This completes the "training" of the RBF network for the XOR problem.

Interpretation of Gaussian Hidden Units

The Gaussian hidden units in an RBF Network are "activated" when the associated regions in the input space are "activated", so they can be interpreted as ***receptive fields***. For each hidden unit there will be a region of the input space that results in an activation above a certain threshold, and that region is the receptive field for that hidden unit. This provides a direct relation to the receptive fields in biological sensory systems.

Another interpretation of the Gaussian RBF is as a ***kernel***. Kernel regression is a general technique for estimating regression functions from noisy data based on the methods of kernel density estimation. The required probability density function, namely $p(y|x)$, can be computed using Bayes law from the probability densities that can be estimated from the training data, namely $p(x)$ and $p(x|y)$. The idea is to write $p(x)$ and $p(x|y)$ as linear combinations of suitable kernels and use the training data to estimate the parameters. Using a Gaussian kernel thus leads to a direct relation between RBF networks and kernel regression. For more details see Haykin-2009 Sections 5.9 and 5.10.

Comparison of RBF Networks with MLPs

When deciding whether to use an RBF network or an MLP, there are several factors to consider. There are clearly similarities between RBF networks and MLPs:

Similarities

1. They are both non-linear feed-forward networks.
2. They are both universal approximators.
3. They can both be used in similar application areas.

It is not surprising, then, to find that there always exists an RBF network capable of accurately mimicking a specific MLP, and vice versa. However the two network types do differ from each other in a number of important respects:

Differences

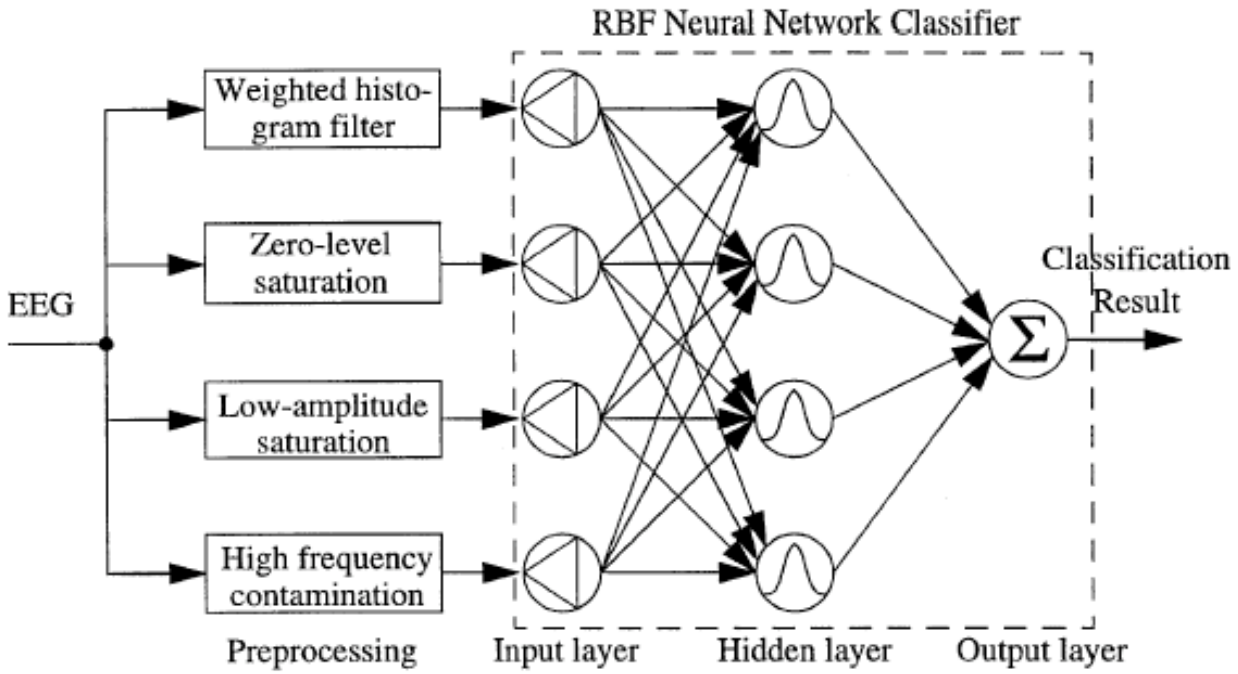
1. RBF networks are naturally fully connected with a single hidden layer, whereas MLPs can have any number of hidden layers and any connectivity pattern.

2. In MLPs, the nodes in different layers share a common neuronal model, though not always the same activation function. In RBF networks, the hidden nodes (i.e., basis functions) have a very different purpose and operation to the output nodes.
3. In RBF networks, the argument of each hidden unit activation function is the ***distance*** between the input and the "weights" (RBF centres), whereas in MLPs it is the ***inner product*** of the input and the weights.
4. RBF networks are usually trained quickly one layer at a time with the first layer unsupervised, which allows them to make good use of unlabelled training data.
5. MLPs are usually trained iteratively with a single global supervised algorithm, which is slow compared to RBF networks and requires many learning parameters to be set well, but allows an early stopping approach to optimizing generalization.
6. MLPs construct ***global*** approximations to non-linear input-output mappings with ***distributed*** hidden representations, whereas RBF networks tend to use ***localised*** non-linearities (Gaussians) at the hidden layer to construct ***local*** approximations.

Generally, for approximating non-linear input-output mappings, RBF networks can be trained much faster, but an MLP may still allow easier optimization of generalization.

Real World Application – EEG Analysis

One successful RBF network detects epileptiform artefacts in EEG recordings:



Full details can be found in the original journal paper: A. Saastamoinen, T. Pietilä, A. Värri, M. Lehtokangas, & J. Saarinen, (1998). Waveform Detection with RBF Network – Application to Automated EEG Analysis. *Neurocomputing*, vol. 20, pp. 1-13.

Overview and Reading

1. We began by looking at regularization approaches for RBF networks.
2. Then we noted the relevance of Cover's theorem on the separability of patterns, and saw how to use RBF networks for classification tasks.
3. As a concrete example, we considered how the XOR problem could be dealt with by an RBF network. We explicitly computed all the basis functions and output weights for such a network.
4. Next we looked at the interpretation of Gaussian hidden units.
5. Then we went through a full comparison of RBF networks and MLPs.
6. We ended by looking at a real world application – EEG analysis.

Reading

1. Bishop: Sections 5.3, 5.4, 5.6, 5.7, 5.8, 5.10
2. Haykin-2009: Sections 5.2, 5.8, 5.9, 5.10, 5.11