Radial Basis Function Networks: Applications

Introduction to Neural Networks : Lecture 14

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

Supervised RBF Network Training

Supervised training of the basis function parameters will generally give better results than unsupervised procedures, but the computational costs are usually enormous.

The obvious approach is to perform gradient descent on a sum squared output error function as we did for our MLPs. The error function would be

$$E = \sum_p \sum_k (y_k(\mathbf{x}^p) - t_k^p)^2 = \sum_p \sum_k (\sum_{j=0}^{M} w_{kj}\phi_j\left(\mathbf{x}^p, \boldsymbol{\mu}_j, \sigma_j\right) - t_k^p)^2$$

and we would iteratively update the weights/basis function parameters using

$$\Delta w_{jk} = -\eta_w \frac{\partial E}{\partial w_{jk}} \qquad \Delta \mu_{ij} = -\eta_\mu \frac{\partial E}{\partial \mu_{ij}} \qquad \Delta \sigma_j = -\eta_\sigma \frac{\partial E}{\partial \sigma_i}$$

We have all the problems of choosing the learning rates η, avoiding local minima and so on, that we had for training MLPs by gradient descent. Also, there is a tendency for the basis function widths to grow large leaving non-localised basis functions.

Regularization Theory for RBF Networks

Instead of restricting the number of hidden units, an alternative approach for preventing overfitting in RBF networks comes from the theory of ***regularization***, which we saw previously was a method of controlling the smoothness of mapping functions.

We can have one basis function centred on each training data point as in the case of exact interpolation, but add an extra term to the error measure which penalizes mappings which are not smooth. If we have network outputs $y_k(\mathbf{x}^p)$ and sum squared error measure, we can introduce some appropriate differential operator $\mathbf{P}$ and write

$$E = \tfrac{1}{2}\sum_p \sum_k (y_k(\mathbf{x}^p) - t_k^p)^2 + \lambda \sum_k \int |\mathbf{P} y_k(\mathbf{x})|^2 \, d\mathbf{x}$$

where λ is the regularization parameter which determines the relative importance of smoothness compared with error. There are many possible forms for $\mathbf{P}$, but the general idea is that mapping functions $y_k(\mathbf{x})$ which have large curvature should have large values of $|\mathbf{P} y_k(\mathbf{x})|^2$ and hence contribute a large penalty in the total error function.

Computing the Regularized Weights

Provided the regularization term is quadratic in $y_k(\mathbf{x})$, the second layer weights can still be found by solving a set of linear equations. For example, the regularizer

$$\lambda \sum_k \left[\sum_p \sum_i \frac{1}{2} \left(\frac{\partial^2 y_k(\mathbf{x}^p)}{\partial x_i^{\,2}} \right)^2 \right]$$

certainly penalizes large output curvature, and minimizing the error function E now leads to linear equations for the output weights that are no harder to solve than we had before

$$\mathbf{M}\mathbf{W}^{\mathsf{T}} - \mathbf{\Phi}^{\mathsf{T}}\mathbf{T} = 0$$

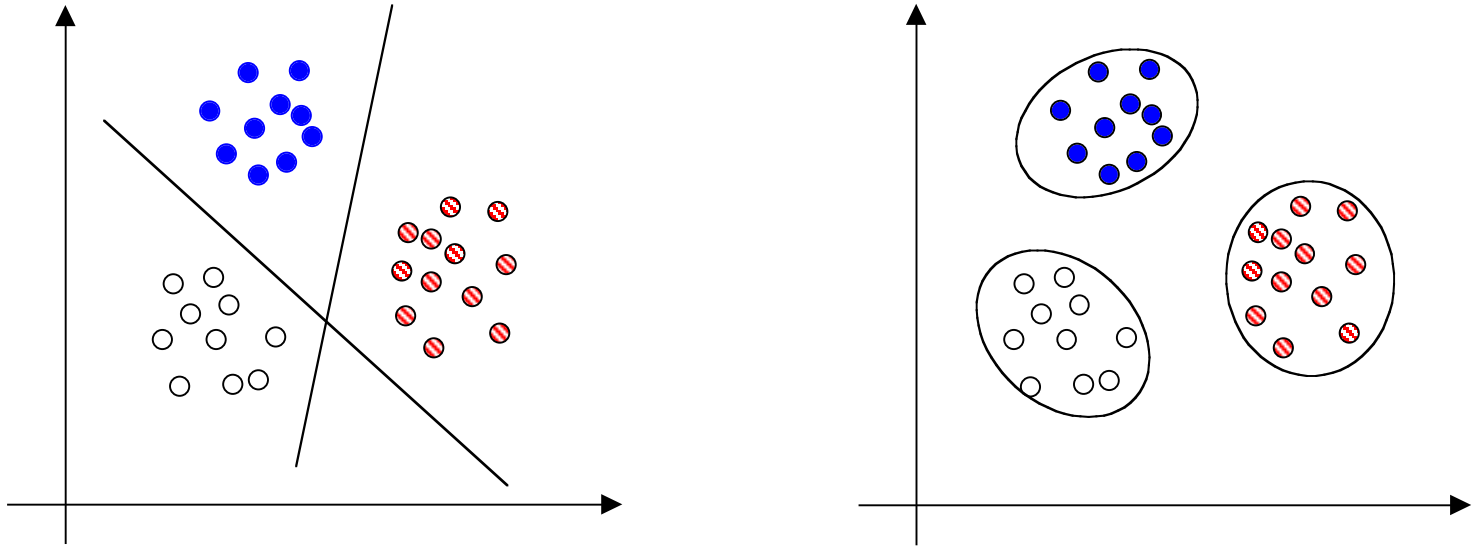
In this we have defined the same matrices with components $(\mathbf{W})_{kj} = w_{kj}$, $(\mathbf{\Phi})_{pj} = \phi_j(\mathbf{x}^p)$, and $(\mathbf{T})_{pk} = \{t_k^p\}$ as before, and also the regularized version of $\mathbf{\Phi}^{\mathsf{T}}\mathbf{\Phi}$

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

Clearly for $\lambda = 0$ this reduces to the un-regularized result we derived in the last lecture.

RBF Networks for Classification

So far we have concentrated on RBF networks for function approximation. 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 have an RBF network perform classification – we simply need to 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.

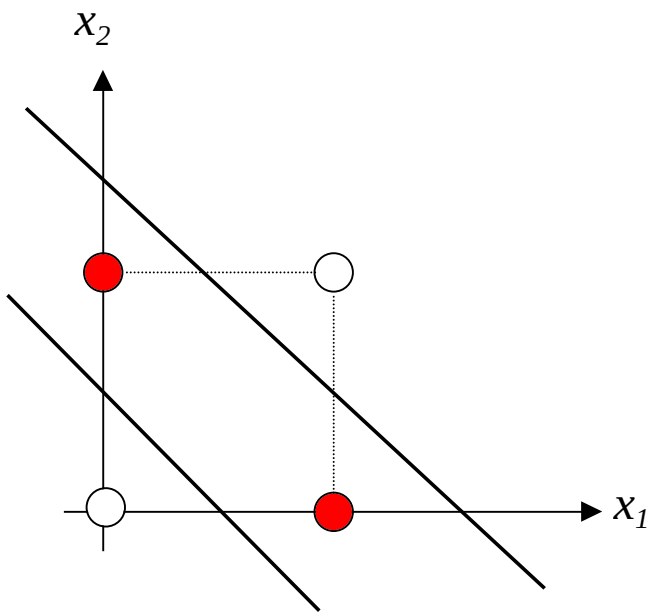
The underlying justification is found in ***Cover's theorem*** which states that "A complex pattern classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low dimensional space". We know that once we have linear separable patterns, the classification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

The XOR Problem Revisited

We are already familiar with the non-linearly separable XOR problem:

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



We know that Single Layer Perceptrons with step or sigmoidal activation functions cannot generate the right outputs, because they are only able to form a single decision boundary. To deal with this problem using Perceptrons we needed to 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:

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

To perform the XOR classification in an RBF network, we start by deciding how many basis functions we need. Given there are four training patterns and two classes, M = 2 seems a reasonable first guess. We then need to decide on the basis function centres. The two separated zero targets seem a good bet, so we can set $\boldsymbol{\mu}_1 = (0,0)$ and $\boldsymbol{\mu}_2 = (1,1)$ and the distance between them is $d_{max} = \sqrt{2}$. We thus have the two basis functions

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

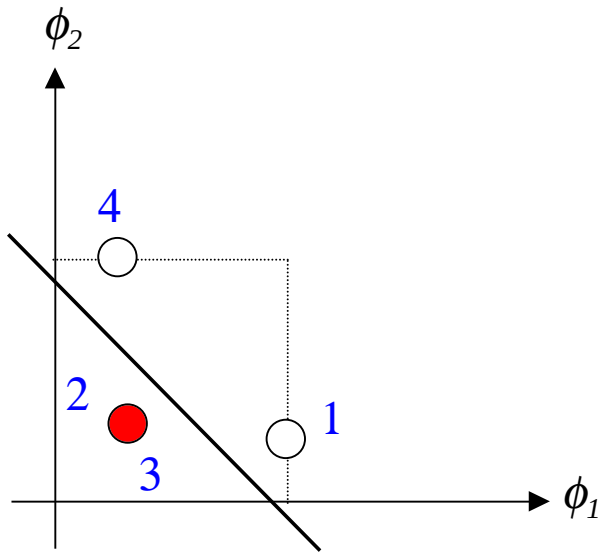
$$\phi_2(\mathbf{x}) = \exp\left(-\left\|\mathbf{x}-\boldsymbol{\mu}_2\right\|^2\right) \qquad \text{with} \quad \boldsymbol{\mu}_2 = (1,1)$$

This will hopefully transform the problem into a linearly separable form.

The XOR Problem Basis Functions

Since the hidden unit activation space is only two dimensional we can easily plot how the 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



We can see that the patterns are now linearly separable. Note that in this case we did not have to increase the dimensionality from the input space to the hidden unit/basis function space – the non-linearity of the mapping was sufficient. **Exercise**: check what happens if you chose two different basis function centres.

The XOR Problem Output Weights

In this case we just have one output $y(\mathbf{x})$, with one weight w_j to each hidden unit j, and one bias $-\theta$. This gives us the network's input-output relation for each input pattern $\mathbf{x}$

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

Then, if we want the outputs $y(\mathbf{x}^p)$ to equal the targets t^p, we get the four equations

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

Three are different, and we have three variables, so we can easily solve them to give

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

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

Comparison of RBF Networks with MLPs

There are clearly a number of similarities between RBF networks and MLPs:

Similarities

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

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

Differences

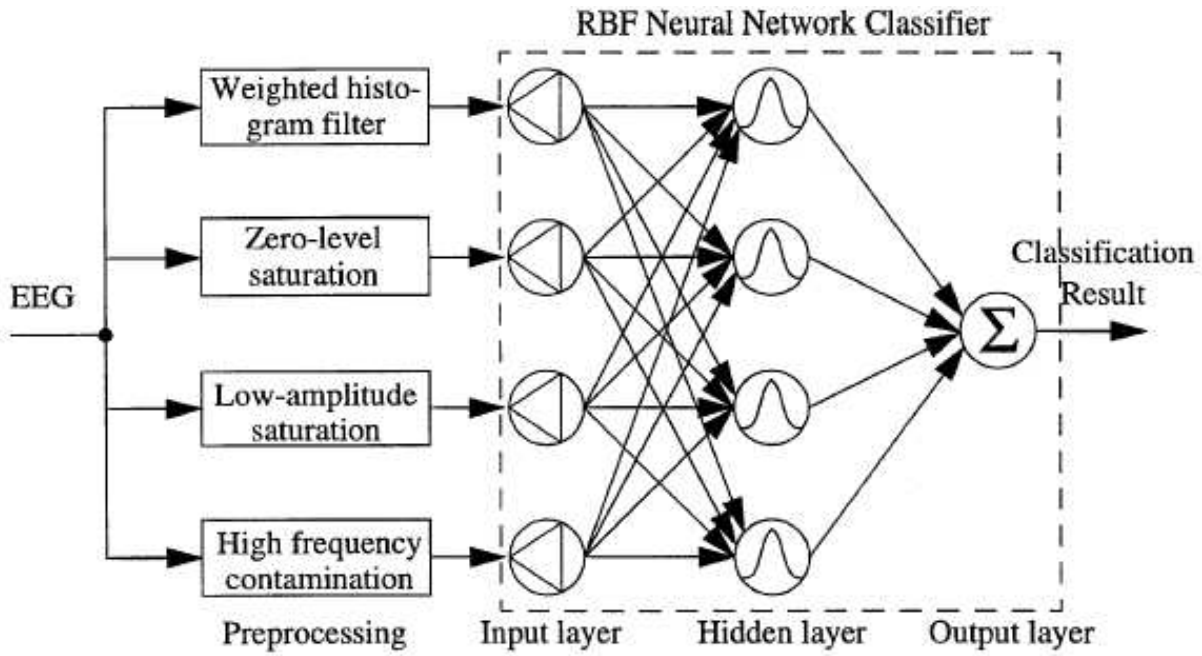
1. An RBF network (in its natural form) has a single hidden layer, whereas MLPs can have any number of hidden layers.

2. RBF networks are usually fully connected, whereas it is common for MLPs to be only partially connected.
3. In MLPs the computation nodes (processing units) in different layers share a common neuronal model, though not necessarily the same activation function. In RBF networks the hidden nodes (basis functions) operate very differently, and have a very different purpose, to the output nodes.
4. 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.
5. MLPs are usually trained with a single global supervised algorithm, whereas RBF networks are usually trained one layer at a time with the first layer unsupervised.
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.

Although, for approximating non-linear input-output mappings, the RBF networks can be trained much faster, MLPs may require a smaller number of parameters.

Real World Application – EEG Analysis

One successful RBF network detects epileptiform artefacts in EEG recordings:



For full details see the original paper by: 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 how supervised RBF training would work.
2. Then we considered using regularization theory for RBF networks.
3. We then saw how we can use RBF networks for classification tasks and noted the relevance of Cover's theorem on the separability of patterns.
4. 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 our network.
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, 5.7, 5.8, 5.10
2. Haykin: Sections 5.2, 5.5, 5.6, 5.11, 5.14, 5.15