Learning from Data
Adaptive Basis Function Networks and Committees

Course lecturer: Amos Storkey
a.storkey@ed.ac.uk
Course page : http://www.anc.ed.ac.uk/∼amos/lfd/

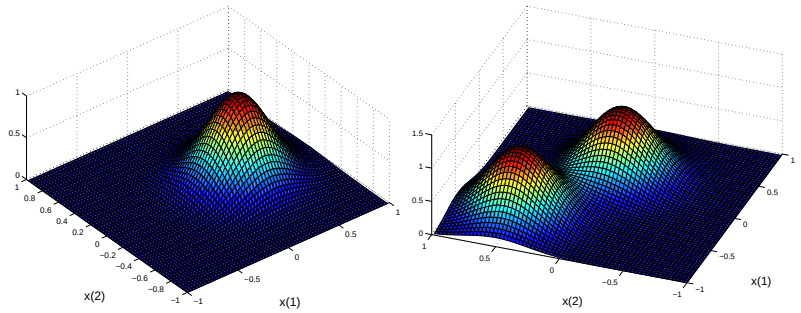

Figure 1: Left: The output of an RBF function $\exp(-\frac{1}{2}\left(\mathbf{x}-\mathbf{m}^1\right)^2/\alpha^2)$. Here $\mathbf{m}^1 = (0, 0.3)^T$ and $\alpha = 0.25$. Right: The combined output for two RBFs, $\mathbf{m}^2 = (0.5, -0.5)^T$.

1 Adaptive Basis Function Networks

Linear weighted inputs

In neural networks, typically the output of each node (or neuron) in the network is some non-linear function of a linear combination of the nodes entering the network (the parents). That is,

$$y_i = g_i\left(\sum_j w_{ij}x_j + \mu_i\right) \tag{1.1}$$

As previously discussed, because the output of the node only depends on a linear combination of the inputs to the network node/neuron, essentially there is only variability in one direction in the input space (where by input I mean the inputs to the node). We can make a bump, but only a one dimensional bump, albeit in a high dimensional space. To get variability in more than one direction, we need to combine neurons together. Since it is quite reasonable to assume that we want variability in many dimensions in the input space, particularly in regions close to the training data, we typically want to make bumps near the data.

1.1 Adaptive Basis Functions

In the case of linear parametric models, we saw how we can approximate a function using a linear combination of fixed basis functions. Localised Radial Basis Functions($\exp(-(\mathbf{x}-\mathbf{m})^2)$) are a reasonable choice for the "bump" function type approach. The output of this function depends on the *distance* between $\mathbf{x}$ and the centre of the RBF $\mathbf{m}$. Hence, in general, the value of the basis function will change as $\mathbf{x}$ moves in any direction, apart from those that leave $\mathbf{x}$ the same distance from $\mathbf{m}$, see fig(1). Previously, we suggested that a good strategy for placing centres of basis functions is to put one on each training point input vector. However, if there are a great number of training patterns, this may not be feasible. Also, we may wish to use the model for compression, and placing a basis function on each training point may not give a particularly high compression. Instead we could adapt the positions of the centres of the basis functions, treating these also as adaptable parameters. In general, an adaptive basis function network is of

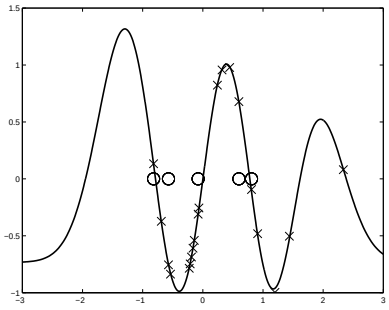

Figure 2: A RBF function using five basis functions. Note how the positions of the basis function centres, given by the circles, are not uniform.

the form

$$y(\mathbf{x}, \boldsymbol{\theta}) = \sum_i w_i \phi_i(\mathbf{x}, \mathbf{b}^i) \tag{1.2}$$

where now each basis function $\phi_i(\mathbf{x}, \mathbf{b}^i)$ has potentially its own parameters that can be adjusted. $\boldsymbol{\theta}$ represents the set of all adjustable parameters. If the basis functions are non-linear, then the overall model is a non-linear function of the parameters.

2 Training Adaptive Basis Functions

Let us consider, for convenience, only a single output variable y. Given a set of input-output pairs, $D = \{(\mathbf{x}^\mu, y^\mu), \mu = 1, \ldots, P\}$, how can we find appropriate paramters $\boldsymbol{\theta}$ that minimise the error that the network makes in fitting this function?

Regression

A suitable choice of energy or error function for regression is

$$E_{train}(\boldsymbol{\theta}) = \sum_\mu \left(y^\mu - f(\mathbf{x}^\mu, \boldsymbol{\theta})\right)^2 \tag{2.1}$$

We can train this network by any standard (non-linear) optimisation algorithm, such as conjugate gradient descent.

However, one should always bear in mind that, in general, the training of complex non-linear models with many parameters is extremely difficult.

Classification

A suitable choice of energy or error function to minimise for classification is the negative log likelihood (if $y^\mu \in \{0, 1\}$)

$$E_{train}(\boldsymbol{\theta}) = -\sum_\mu \left(y^\mu \log f^\mu + (1 - y^\mu) \log(1 - f^\mu)\right) \tag{2.2}$$

where $f^\mu = f(\mathbf{x}^\mu, \boldsymbol{\theta})$.

Regularisation

The smoothness of the RBF mapping is mainly determined by the width of the basis functions. The easiest approach is to use a validation set to determine α and not to regularise any of the other parameters.

Initialisation

The quality of the solutions is critically dependent on the initial parame-

ter settings, in particular where we initially speculatively place the basis function centres.

One reasonable initialisation strategy is to place the centres on a randomly chosen subset of the data, and then solve for the hidden to output weights $\mathbf{w}$ easily (this is just a linearised parameter model if we consider the basis functions fixed).

Another approach is to use K-means clustering (see later chapter) to set the centres of the basis functions. Given the initial centres of the basis functions, we can solve for the weights easily. This gives an initial setting for the basis function and weight values.

Optimisation Strategies

Perhaps the most obvious thing to do is to treat both the weights $\mathbf{w}$ and basis function parameters $\mathbf{b}^i$ together as one parameter set, $\boldsymbol{\theta}$, and optimise the objective function with respect to $\boldsymbol{\theta}$. Example code for regression using a this approach is given below. It is straightforward to adapt this for classification. This code is not fully vectorised for clarity, and also uses the `scg.m` function, part of the NETLAB (see `http://www.ncrg.aston.ac.uk`) package.

```
% adaptive RBF for regression :
% Training data: (each row contains a datapoint)
x=randn(20,2); % two dimensional inputs
y = sin(4*sum(x,2)); % one dimensional ouputs
n = size(x,2); p = size(x,1);

K = 5; % number of basis functions
r = randperm(p); w = x(r(1:K),:)'; % initialise centres to random training points
v = randn(K,1); b0=1; % other initial parameters

nw = prod(size(w)); % number of weight parameters
th_init=[reshape(w,1,nw),v',b0];  % initial parameter vector
alpha = 0.5; % basis function width

% now use Scaled Conjugate Gradients to find optimal parameters:
options = zeros(1,18); options(9)=1; options(1)=1;options(14)=200;
[th_opt]=scg('E_rbf',th_init,options,'grad_E_rbf',x,y,nw,K,alpha);
train_err = sum( (rbfn(x,th_opt,nw,K,alpha) - y).^2)

function [f,w,v,b0,a,h] = rbfn(x,th,nw,K,alpha)
  p = size(x,1); n = size(x,2);
  w=reshape(th(1:nw),n,K); function [f,w,v,b0,a,h] = rbfn(x,th,nw,K,alpha)
  p = size(x,1); n = size(x,2);
  w=reshape(th(1:nw),n,K);   % get the parameters from vector th
  v=th(nw+1:nw+K)';  b0 =th(nw+K+1);
  a = sqdist(x,w'); % find (x-w)^2
  h = exp(-0.5*a./alpha^2); % RBF hidden units
  a0 = h*v + repmat(b0,p,1); % output activation
  f = a0; % output transfer function is the identity

function g = grad_rbfn(x,th,nw,K,alpha)
  p = size(x,1); n=size(x,2);
  [f,w,v,b0,a,h] =rbfn(x,th,nw,K,alpha);

   for mu =1:p % Do this using loops for clarity :
     for i = 1:size(w,2) % done using loops for clarity
       gw_tmp(:,i) = v(i)*(x(mu,:)' - w(:,i)).*h(mu,i)./alpha^2;
     end
%     gw_tmp = v'.*(repmat(x(mu,:)',1,size(w,2))-w).*h(mu,:)./alpha^2; %loopless
     gw(mu,:) = reshape(gw_tmp,1,prod(size(w)));
   end
  gv=h; gb0=ones(p,1); g = [gw gv gb0];

function E = E_rbf(th,x,y,nw,K,alpha)
[f,w,v,v0]=rbfn(x,th,nw,K,alpha);  E = sum((y-f).^2)

function gE = grad_E_rbf(th,x,y,nw,K,alpha)
  [f,w,v,b0] = rbfn(x,th,nw,K,alpha);  gf = grad_rbfn(x,th,nw,K,alpha);
  gE = 2*sum(repmat(f-y,1,length(th)).*gf);

function n2 = sqdist(x, c)
%SQDIST Calculates squared distance between two sets of points.
%
%   Description
%   D = SQDIST(X, C) takes two matrices of vectors and calculates the
%   squared Euclidean distance between them.  Both matrices must be of
%   the same column dimension.  If X has M rows and N columns, and C has
%   L rows and N columns, then the result has M rows and L columns.  The
%   I, Jth entry is the  squared distance from the Ith row of X to the
%   Jth row of C.

[ndata, dimx] = size(x); [ncentres, dimc] = size(c);
if dimx~=dimc
    error('Data dimension does not match dimension of centres')
end
n2=repmat((sum((x.^2)',1))',1,ncentres)+repmat(sum((c.^2)',1),ndata,1)-2.*(x*(c'));
% Rounding errors occasionally cause negative entries in n2
if any(any(n2<0)); n2(n2<0) = 0; end
```

An example is given in fig(2) where we see that the optimal solution (as found by the optimisation algorithm) produces a non-uniform placing of the basis function centres. However, there is another strategy which, in practice, may be preferable:

1. For fixed basis function parameters $\mathbf{b}^i$, find the best weights $\mathbf{w}$ (this is easy to do since this is just corresponds to solving a linear system).
2. For fixed weights $\mathbf{w}$, find the best basis function parameters. (This is the difficult step since there will typically be many basis function parameters, and the objective function depends in a highly non-linear way on the basis function parameters).

We can iterate these two stages to improve the solution. The slight practical advantage of this is that the parameter space in which we search for a solution to a non-linear optimisation problem is slightly reduced since we only optimise with respect to the $\mathbf{b}^i$ parameters.

2.1 Non-local Basis Functions

If we use basis functions that decay rapidly from a 'centre', as in the case $\exp(-(\mathbf{x}-\mathbf{m})^2)$, the basis function value will always decay to zero once we are far away from the training data. In the case of binary classification and a logistic sigmoid for the class output, this may be reasonable since we would then predict any new datapoint far away from the training data with a complete lack of confidence, and any assignment would be essentially random. However, in regression, using say a linear combination of basis function outputs would always give zero far from the training data. This may give the erroneous impression that we are therefore extremely confident that we should predict an output of zero far away from the training data whilst, in realilty, this is simply an artefact of our model. For this reason, it is sometimes preferable to use basis functions that are non-local – that is, they have appreciable value over all space, for example, $(\mathbf{x}-\mathbf{m})^2 log((\mathbf{x}-\mathbf{m})^2)$. Whilst any single output will tend to infinity away from the training data, this serves to remind the user that, far from the training data, we should be wary of our predictions.

3 Committees

Drawbacks of the non-linear approaches we have looked at – neural networks and their cousins adaptive basis functions – are

1. Highly complex energy/error surfaces give rise to multiple solutions since global optima are impossible to find.
2. We have no sense of the confidence in the predictions we make (particularly in regression).

Whilst there are alternative (and in my view more attractive) approaches around these problems, we can exploit the variability in the solutions found to produce a measure of confidence in our predictions. The idea is to form a committee of networks from the solutions found. For example, for regression, we could train (say) M networks on the data and get M different parameter

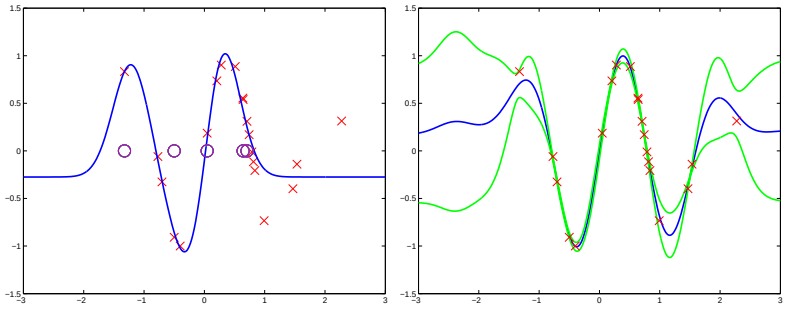

Figure 3: Left: A single solution using Adaptive Basis Functions to fitting the training data (crosses). The centres of the five basis functions are given by the circles. Right: A committee prediction from six individual solutions of the form given on the left. The central line is the average prediction – note how this still decays to zero away from the training data. The lines around the central line a one standard deviation confidence intervals.

solutions $\boldsymbol{\theta}^1, \ldots, \boldsymbol{\theta}^M$. The average network function would then be

$$\bar{f}(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}, \boldsymbol{\theta}^i). \tag{3.1}$$

A measure of the variability in the predictions is given by the variance :

$$\mathrm{var}(f)(\mathbf{x}) = \frac{1}{M} \sum_{i=1}^{M} (f(\mathbf{x}, \boldsymbol{\theta}^i) - \bar{f}(\mathbf{x}))^2. \tag{3.2}$$

A useful plot of confidence in our predictions is then to use one standard deviation error bars :

$$\bar{f}(\mathbf{x}) \pm \sqrt{\mathrm{var}(f)(\mathbf{x})} \tag{3.3}$$

In fig(3) we give an example using a committee of six adaptive basis functions.

The committee idea is quite general and applicable to any model. Whilst this approach is rather heuristic and leaves some questions unanswered (why did we choose uniform weighting of the solutions for example) it is nevertheless an intuitive an reasonably robust way of gaining confidence in model predictions (these issues can be solved by a Bayesian treatment beyond the scope of this course). Note that the committee approach does not necessarily solve the issue of over confident regression predictions away from training data. As seen in fig(3) both the mean and confidence will collapse around zero as we move far from the training data. This is a good reason to use non-local basis functions in this case since typically, the variability will become unbounded as we move away from the training data.