LECTURE 4: HIERARCHICAL SYSTEMS

Why does a linear discriminant (LD) fail for the XOR problem? Of course, because the problem requires *two* decision boundaries. How about having two linear discriminators, each working on half of the problem with another LD analyser to decide which of these to use? The latter acts as a binary or gating switch. This is shown in Figure 1.

 Make sure you understand how this works.

Of course, we are free to use logistic discriminators if we wish. Recall that the

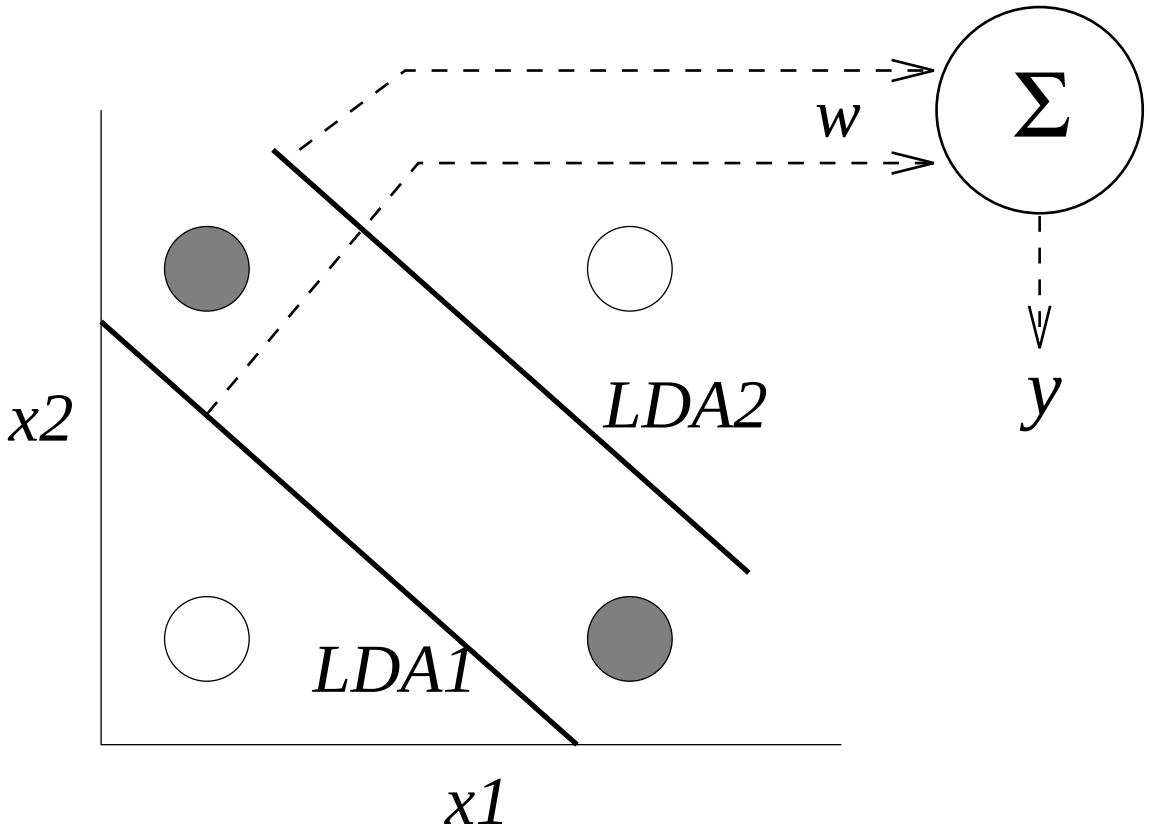

Figure 1: Two LDAs and a gating LDA for the XOR problem.

logistic discriminator (and sometimes the LDA) were referred to as the *perceptron* in the late 1950s and excitement was generated due to the apparent similarity to a 'neuron'. The basic system of Figure 1 was hence referred to as the 'multi-layer perceptron' (MLP). It was realised, furthermore, that the second layer did not need to just be a binary gate. We hence generalise the MLP structure in Figure 2. This structure can be modelled by the functional form:

$$y = f_2[f_1(x)]$$

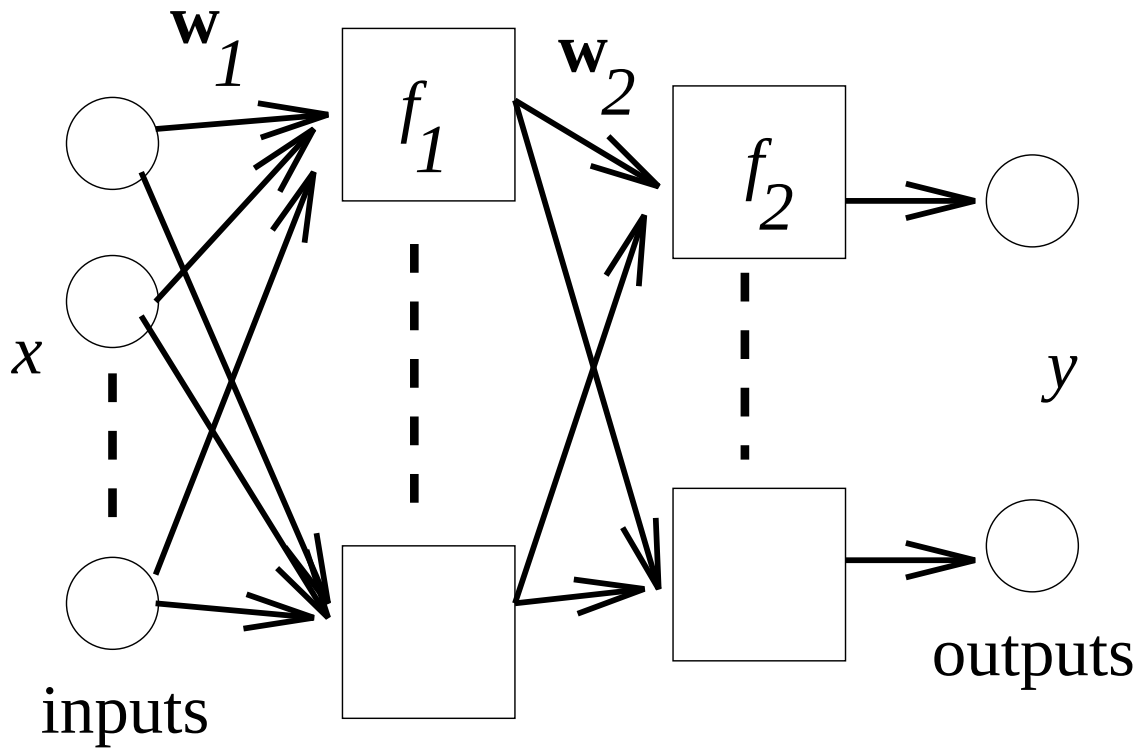

Figure 2: The MLP

where $f_1(x)$ is a transform of the input to a so-called *hidden* space and f_2 is a transform from this hidden space (or layer) to the output y.

This chapter will look, in part, in more detail at this system.

The MLP

The form of the transform function $f_1(x)$ may be seen as that of defining candidate decision boundaries in the input space. This means that this function can have the form of a step function acting on a weighted combination of the inputs (with an additional bias term). A step function such as in Figure 3(a) can be used. Indeed this makes a two-layer system which works! The trouble is that the function is non-differentiable, which means that we cannot use an efficient optimisation method. A function such as the sigmoid (logistic) may just as easily be used as in Figure 3(b) which is smooth.

What about the final function $f_2[.]$? We need to discuss a little more about *error functionals* beforehand.

Error functionals revisited

We have already looked at the case in which we assume that errors in the output, $y - t$, are Gaussian distributed. This leads naturally to the sum-of-squares (SSE)

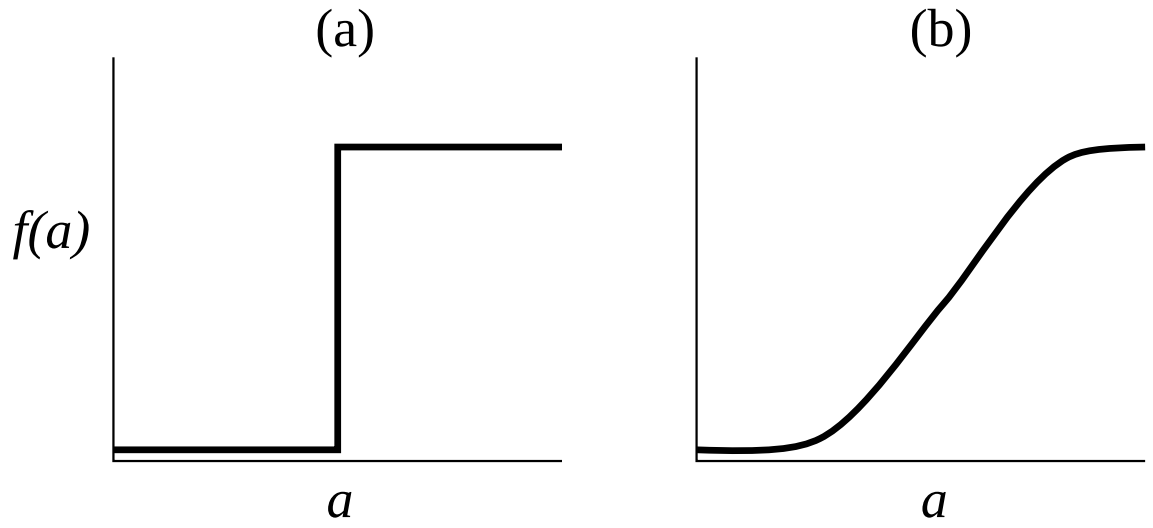

Figure 3: (a) Step function (hard-limiter) and (b) sigmoid.

error function.

Writing

$$P(T \mid X) = \prod_{n=1}^{N} p(t^n \mid x^n)$$
$$\propto \exp\left\{ -\sum_{n=1}^{N} \frac{\beta}{2}(y^n - t^n)^2 \right\}$$

Remember that the error function is the negative log of the density function of the output errors (high uncertainty leads to high errors), so the SSE error is (ignoring constant terms):

$$E_{SSE} = \frac{1}{2}\sum_{n=1}^{N}(y^n - t^n)^2$$

If we are to optimise the set of parameters (weights) which make up the network then, during training, the error must be capable of being fed back into the structure so as to adapt the weights (just a standard pre-requisite of adaptive systems). If

$$y = f_2[\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}] = f_2[a]$$

in which $\boldsymbol{\phi}$ is the output of responses from the *hidden layer*, somtimes referred to as a *latent space*, and $\mathbf{w}$ are adaptive parameters then:

$$\frac{\partial E}{\partial a} = \frac{\partial y}{\partial a}\frac{\partial E}{\partial y} = \frac{\partial y}{\partial a}(y - t)$$

if we desire $\frac{\partial E}{\partial a} \propto (y - t)$ then $f_2[a] \propto a$ i.e. *the connectivity from hidden to output layers is <u>linear</u>*. For regression, therefore, we would use the MLP with f_1

as a sigmoidal function (such as the logistic or tanh functions) and f_2 as linear. We can regard this form then as a set of logistic discriminators with a nesting of linear discriminators on top.

The errors being Gaussian distributed around the output is the appropriate model for *regression*, but certainly not for *classification*.

We will consider, to begin with, just two classes, C_1 and C_2. If $P(C_1 \mid x) = y$ then $P(C_2 \mid x) = 1 - y$. The target coding in our labelled training set consists of $t = 1$ if x belongs to C_1 and $t = 0$ if $x \in C_2$. The probability density is Bernoulli of the form:

$$P(t \mid x) = y^t(1-y)^{1-t}$$

and for N data,

$$P(T \mid X) = \prod_{n=1}^{N}(y^n)^{t^n}(1-y^n)^{1-t^n}$$

Taking the negative log of the density function gives the *cross entropy error function*,

$$E_{XE} = -\sum_{n=1}^{N}\{t^n \ln y^n + (1-t^n)\ln(1-y^n)\}$$

Consider now just the n-th pattern's contribution to this error (dropping the superscript to make it less messy...),

$$E = -\left(t \ln y + (1-t)\ln(1-y)\right)$$

Consider again the feedback error,

$$\frac{\partial E}{\partial a} = \frac{\partial y}{\partial a}\frac{\partial E}{\partial y}$$

If we consider the output in the form

$$y = g(a)$$

where $g(.)$ is some smooth, differentiable function then,

$$\frac{\partial E}{\partial a} = g'(a)\frac{(y-t)}{y(1-y)}$$

Once again we desire this gradient feedback error to be of the form $(y - t)$ hence

$$g'(a) = g(a)[1 - g(a)]$$

which is satisfied by

$$g(a) = \frac{1}{1+\exp(-a)}$$

i.e. just the logistic sigmoid function we have already come across.

If the training set is large then we may replace the summation in the error equation by a integrals over t and x,

$$\begin{aligned} E &= -\int\int \{t \ln y(x) + (1-t)\ln(1-y(x))\} P(t \mid x) P(x) dt dx \\ &= -\int \{\langle t \mid x\rangle \ln y(x) + (1 - \langle t \mid x\rangle)\ln(1-y(x))\} P(x) dx \end{aligned}$$

which is minimised when

$$y(x) = \langle t \mid x\rangle = \int t P(t \mid x) dt$$

which is just as in the case of regression. The best estimate is the one which estimates the *conditional average* of the targets on the input.

For a 2-class coding,

$$P(t \mid x) = \delta(t-1) P(C_1 \mid x) + \delta(t) P(C_2 \mid x)$$

giving

$$y(x) = \langle t \mid x\rangle = P(C_1 \mid x)$$

so confirming that the best estimate is also the posterior probability.

Multiple output classes

So far we have only looked at the 2-class case. If there are K output classes, $\{C_1, ..., C_K\}$, then an entropy error of the form

$$E = -\sum_n \sum_{k=1}^{K} t_k^n \ln y_k^n$$

may be used. This has a minimum when $y = t$ and hence we may use an error function relative to this minimum,

$$E = \sum_n \sum_k t_k^n \ln\left(\frac{y_k^n}{t_k^n}\right)$$

which is zero iff[1] $y = t$. Once more we want the feedback gradient of the error to be of the form $(y - t)$. If

$$y_k = g(a_1, ..., a_K)$$

then $g(.)$ is the *softmax* or *generalised logistic* function,

$$y_k = g(a_1, ..., a_K) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$

[1] 'iff' means 'if and only if'.

Again just consider (for notational ease) the n-th pattern,

$$\frac{\partial E}{\partial a_k} = \sum_{k'} \frac{\partial E}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial a_k}$$

which for the softmax and error functions defined above,

$$\frac{\partial y_{k'}}{\partial a_k} = y_{k'}\delta_{kk'} - y_{k'}y_k$$

and

$$\frac{\partial E}{\partial y_{k'}} = -\frac{t_k}{y_{k'}}$$

Putting this together gives

$$\frac{\partial E}{\partial a_k} = y_k - t_k$$

just what we want!

In conclusion then, the input to hidden function mapping in an MLP consists of logistic discriminators,

$$\phi_j = g(\mathbf{w}_{1,j}^{\mathsf{T}}\mathbf{x} + w_{bias1,j})$$

and the hidden to output mapping of

Classification: Another layer of generalised logistic discriminators (which for more that 2 classes gives the *softmax* function) of the form

$$y_k = f_2(\mathbf{w}_{2,k}^{\mathsf{T}}\boldsymbol{\phi} + w_{bias2,k})$$

Regression: The functional $f_2(.)$ in the above is just the linear function, i.e. $f_2(a) = a$. The free parameters in the MLP are the first and second layer weights and biases.

The Radial-Basis Function (RBF) approach

The other major feed-forward methodology is that of the RBF. We have seen in the previous chapter how a set of kernel set of density estimators gave rise to a method for estimating outputs both for regression and classification. The form of the RBF system is a non-linear transform of the input variables of the form:

$$\boldsymbol{\Phi} = (\phi_1(\mathbf{x}), ..., \phi_J(\mathbf{x}))^{\mathsf{T}}$$

following by a transform from this space to the set of outputs,

$$y_k(\mathbf{x}) = f_2(\mathbf{w}_k^{\mathsf{T}}\boldsymbol{\Phi} + w_{bias,k})$$

where f_2 is either linear (regression) or a logistic (incl. softmax) for classification.

The non-linear transform ϕ is chosen to be a function with good analytic properties (and also a universal approximator) and Gaussians and thin-plate splines are the most popular:

$$\phi_{\text{Gaussian}}(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^{\mathsf{T}}\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

where the centre location $\mathbf{m}$ and covariance $\mathbf{C}$ are free parameters.

$$\phi_{\text{spline}}(\mathbf{x}) = r^2 \ln r$$

where $r = \| \mathbf{x} - \mathbf{m} \|$ and $\mathbf{m}$ is a centre location and is a free parameter.

The free parameters in an RBF are hence the parameters of the kernel functions and the weights $\mathbf{w}$ (incl. biases) which couple the kernel functions to the outputs.

Figure 4: Gaussian (a) and thin-plate spline (b) functions in 2-D.

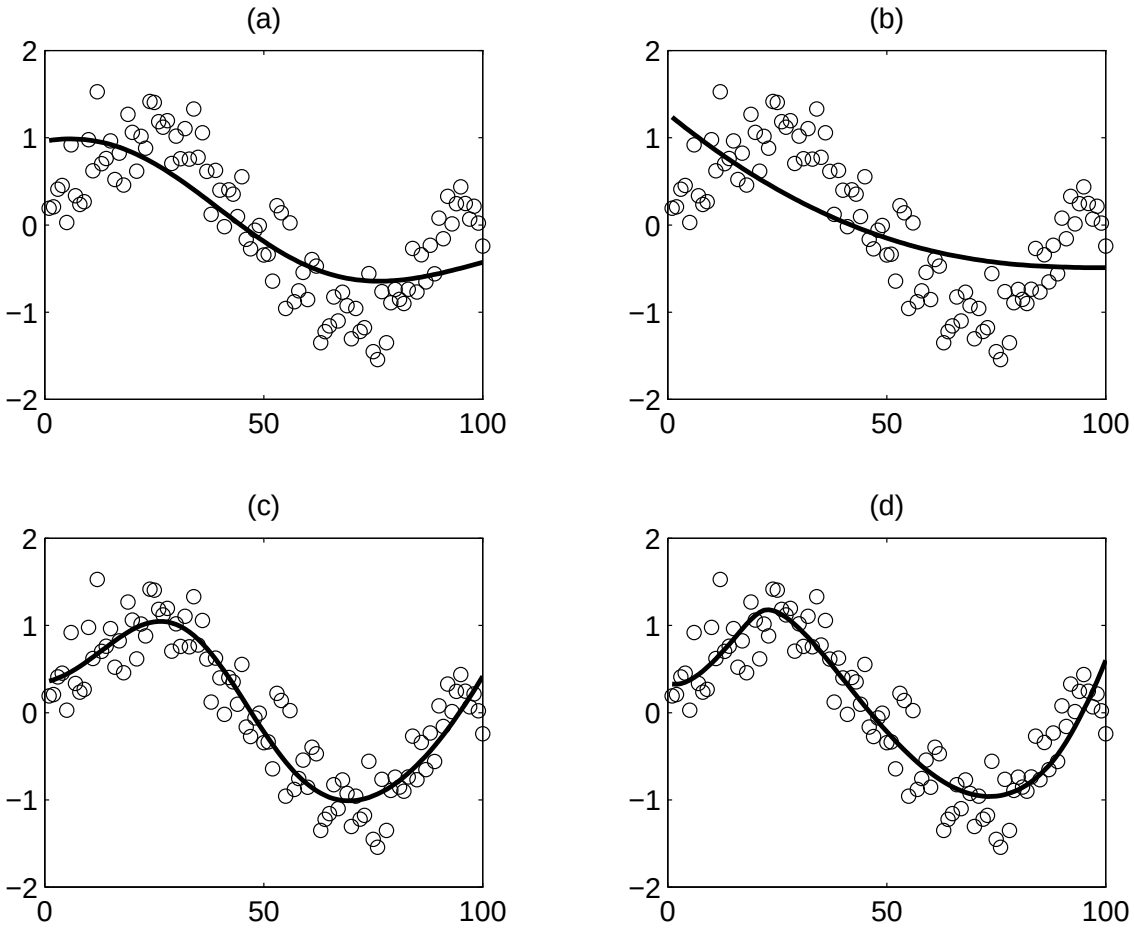

Figure 5: RBF solutions in a regression problem using 2 and 5 kernel functions (a) & (b) and (c) & (d) respectively and Gaussian kernels (a) & (c) or thin-plate splines (b) & (d).

The MLP & RBF - differences in learning

The MLP, remember, has two sets of weight parameters which need to be optimised. We will see later on how this is achieved. The RBF has two sets of parameters, but they are not both 'weights'. The first set parameterise the non-linear mapping functions ϕ and the second are 'weights' which couple the kernels to the outputs.

Two-stage (quick) learning in an RBF

Whilst in the MLP the parameters must be set using *supervised* learning, if we choose the kernels in an RBF system to form a density estimator, then we may break the learning in an RBF down into two stages:

1. learning of the parameters of the kernels and
2. learning the coupling weights to the output.

Stage (1) requires just the input data, X, not any targets. Stage two is *supervised* in that it requires input-target pairs, (x, t). If the problem is a regression one then the

output is simply given as (taking the bias weight into the vector product)

$$y(\mathbf{x}) = \mathbf{w}^{\mathsf{T}} \mathbf{\Phi}(\mathbf{x})$$

which means that, if we can get $\mathbf{\Phi}$ then $\mathbf{w}$ can be obtained from fast pseudo-inverse methods (see Chapter 3). There are two ways in which the set of $\mathbf{\Phi}$ are normally obtained.

1. Choose each kernel (Gaussian) at random or put one on each datum, or
2. optimise a (smaller) set of kernels (Gaussians) to fit the data density $P(\mathbf{x})$.

When discussing generalisation we saw that the more parameters (kernels) the more complex functions could be fitted. Too many kernels and we overfit and learn noise as well as the functions we want to estimate. Scheme (2) has the advantage that, if only a few kernels are needed, they will be placed in reasonable positions and have reasonable widths. We will look in the next Chapter at how to optimise such a set (of Gaussians). Scheme (1) has the advantage of speed but the problems that:

- if only a few kernels are wanted, randomly placing them may be bad, and
- having one on every datum is a recipe for overfitting (we come back to this point in a discussion of *regularisation* later in this Chapter).

The full optimisation method

Of course, all the parameters of the MLP and the RBF may be optmised by adapting them to minimise the appropriate error function. This is *fully supervised* learning. To perform optimisation the gradient of the error function with respect to all the free parameters must be calculated. For all the systems we have looked at this can be done analytically (see Exercises 4).

Regularisation - a brief introduction

We will come back to the issues of regularisation later on. For now we will just look at the basic idea.

When we look at a system that generalises poorly we see that the output is too flexible, it bends to fit noise rather than *smoothly* passing throught the data. This over-curvature can be defined via the second derivative term of the input-output mapping. A simple *regularised* error term is one of the form

$$E_{reg} = E_{data} + \alpha \left\langle \left| \frac{\partial^2 y}{\partial x^2} \right| \right\rangle$$

where E_{data} is the data-dependent error term such as the sum-of-squares functional. Let's just consider a simple RBF network, with a single Gaussian basis function with variance given by σ^2,

$$y = w\phi(x) + w_{bias}$$

for which,

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial \phi}\frac{\partial \phi}{\partial x} = -\frac{w}{\sigma^2}(x-m)\phi(x)$$

hence

$$\frac{\partial^2 y}{\partial x^2} = -\frac{w}{\sigma^2}\phi(x) + \frac{w}{\sigma^4}(x-m)^2\phi(x)$$

when we take expectations over the data set the $(x-m)^2$ term is close to $N\sigma^2$ where N is the number of data points. If this is large then the second term in the Equation dominates and

$$\left\langle \left| \frac{\partial^2 y}{\partial x^2} \right| \right\rangle \approx \frac{N|w|\phi(x)}{\sigma^2}$$

where the absolute value bars $|.|$ are only needed around the weight term as all others are strictly non-negative. We can write $|w| = \sqrt{w^2}$ as well. We see that the regularised error function will be made smaller (better solutions) if the second derivative term gets smaller. This means that smaller values of w^2 and larger values of σ^2 are favoured. Note also that it (intuitively) says that a smaller number of points gives rise to a smoother solution. We will come back to the issue of the w^2 term later on. Figure 6 shows a 50-kernel RBF with changing values of σ^2.

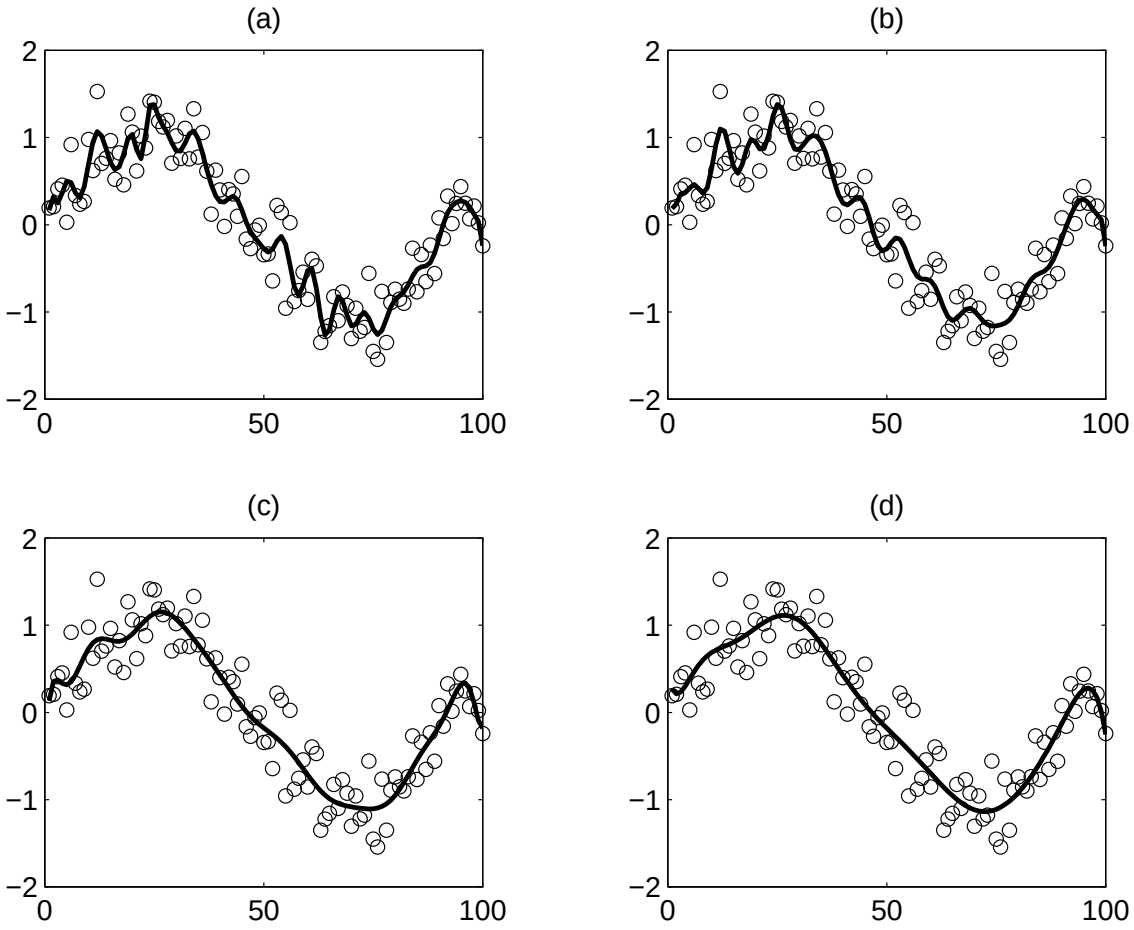

Figure 6: RBF solutions in a regression problem using 50 Gaussian kernel functions and covariances $\mathbf{C} = \sigma^2 \mathbf{C}_{fixed}$. (a-d) $\sigma^2 = 1, 20, 100, 1000$.