3 Nonlinear Regression

Sometimes linear models are not sufficient to capture the real-world phenomena, and thus nonlinear models are necessary. In regression, all such models will have the same basic form, i.e.,

$$\mathbf{y} = f(\mathbf{x}) \tag{1}$$

In linear regression, we have $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$; the parameters $\mathbf{W}$ and $\mathbf{b}$ must be fit to data.

What nonlinear function do we choose? In principle, $f(\mathbf{x})$ could be anything: it could involve linear functions, sines and cosines, summations, and so on. However, the form we choose will make a big difference on the effectiveness of the regression: a more general model will require more data to fit, and different models are more appropriate for different problems. Ideally, the form of the model would be matched exactly to the underlying phenomenon. If we're modeling a linear process, we'd use a linear regression; if we were modeling a physical process, we could, in principle, model $f(\mathbf{x})$ by the equations of physics.

In many situations, we do not know much about the underlying nature of the process being modeled, or else modeling it precisely is too difficult. In these cases, we typically turn to a few models in machine learning that are widely-used and quite effective for many problems. These methods include basis function regression (including Radial Basis Functions), Artificial Neural Networks, and k-Nearest Neighbors.

There is one other important choice to be made, namely, the choice of objective function for learning, or, equivalently, the underlying noise model. In this section we extend the LS estimators introduced in the previous chapter to include one or more terms to encourage smoothness in the estimated models. It is hoped that smoother models will tend to overfit the training data less and therefore generalize somewhat better.

3.1 Basis function regression

A common choice for the function $f(\mathbf{x})$ is a basis function representation[1]:

$$y = f(x) = \sum_k w_k b_k(x) \tag{2}$$

for the 1D case. The functions $b_k(x)$ are called basis functions. Often it will be convenient to express this model in vector form, for which we define $\mathbf{b}(x) = [b_1(x), \ldots, b_M(x)]^T$ and $\mathbf{w} = [w_1, \ldots, w_M]^T$ where M is the number of basis functions. We can then rewrite the model as

$$y = f(x) = \mathbf{b}(x)^T \mathbf{w} \tag{3}$$

Two common choices of basis functions are **polynomials** and **Radial Basis Functions (RBF)**. A simple, common basis for polynomials are the **monomials**, i.e.,

$$b_0(x) = 1, \quad b_1(x) = x, \quad b_2(x) = x^2, \quad b_3(x) = x^3, \quad \ldots \tag{4}$$

[1] In the machine learning and statistics literature, these representations are often referred to as linear regression, since they are linear functions of the "features" $b_k(x)$

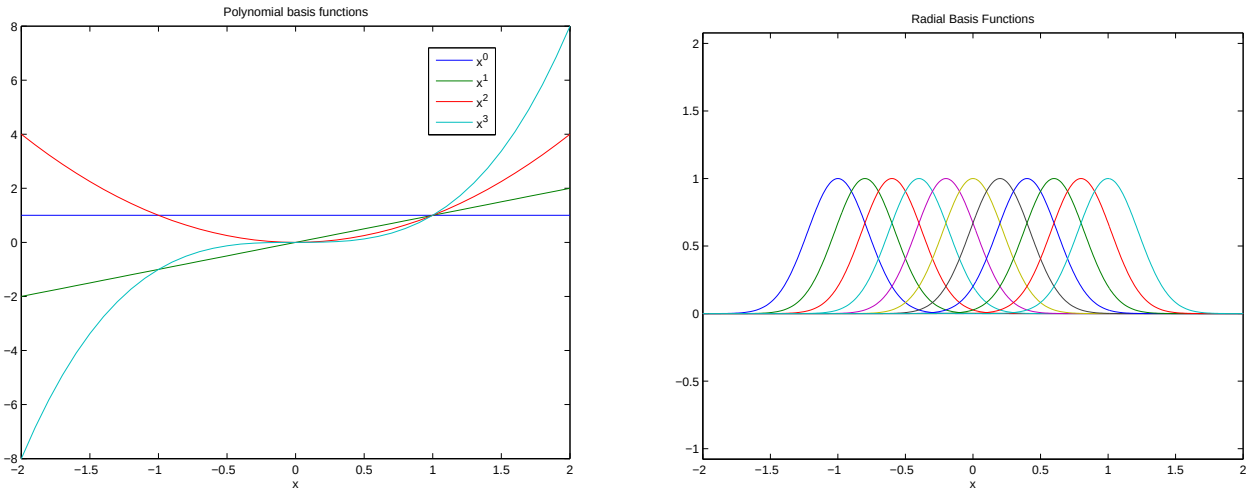

Figure 1: The first three basis functions of a polynomial basis, and Radial Basis Functions

With a monomial basis, the regression model has the form

$$f(x) = \sum w_k x^k \,, \tag{5}$$

Radial Basis Functions, and the resulting regression model are given by

$$b_k(x) = e^{-\frac{(x-c_k)^2}{2\sigma^2}} \,, \tag{6}$$

$$f(x) = \sum w_k e^{-\frac{(x-c_k)^2}{2\sigma^2}} \,, \tag{7}$$

where c_k is the **center** (i.e., the location) of the basis function and σ^2 determines the **width** of the basis function. Both of these are parameters of the model that must be determined somehow.

In practice there are many other possible choices for basis functions, including sinusoidal functions, and other types of polynomials. Also, basis functions from different families, such as monomials and RBFs, can be combined. We might, for example, form a basis using the first few polynomials and a collection of RBFs. In general we ideally want to choose a family of basis functions such that we get a good fit to the data with a small basis set so that the number of weights to be estimated is not too large.

To fit these models, we can again use least-squares regression, by minimizing the sum of squared residual error between model predictions and the training data outputs:

$$E(\mathbf{w}) = \sum_i (y_i - f(x_i))^2 = \sum_i \left(y_i - \sum_k w_k b_k(x) \right)^2 \tag{8}$$

To minimize this function with respect to $\mathbf{w}$, we note that this objective function has the same form as that for linear regression in the previous chapter, except that the inputs are now the $b_k(x)$ values.

In particular, E is still quadratic in the weights $\mathbf{w}$, and hence the weights $\mathbf{w}$ can be estimated the same way. That is, we can rewrite the objective function in matrix-vector form to produce

$$E(\mathbf{w}) = ||\mathbf{y} - \mathbf{B}\mathbf{w}||^2 \tag{9}$$

where $||\cdot||$ denotes the Euclidean norm, and the elements of the matrix $\mathbf{B}$ are given by $\mathbf{B}_{i,j} = b_j(x_i)$ (for row i and column j). In Matlab the least-squares estimate can be computed as $\mathbf{w}^* = \mathbf{B}\backslash\mathbf{y}$.

Picking the other parameters. The positions of the centers and the widths of the RBF basis functions cannot be solved directly for in closed form. So we need some other criteria to select them. If we optimize these parameters for the squared-error, then we will end up with one basis center at each data point, and with tiny width that exactly fit the data. This is a problem as such a model will not usually provide good predictions for inputs other than those in the training set.

The following heuristics instead are commonly used to determine these parameters without overfitting the training data. To pick the basis centers:

1. Place the centers uniformly spaced in the region containing the data. This is quite simple, but can lead to empty regions with basis functions, and will have an impractical number of data points in higher-dimensinal input spaces.

2. Place one center at each data point. This is used more often, since it limits the number of centers needed, although it can also be expensive if the number of data points is large.

3. Cluster the data, and use one center for each cluster. We will cover clustering methods later in the course.

To pick the width parameter:

1. Manually try different values of the width and pick the best by trial-and-error.

2. Use the average squared distances (or median distances) to neighboring centers, scaled by a constant, to be the width. This approach also allows you to use different widths for different basis functions, and it allows the basis functions to be spaced non-uniformly.

In later chapters we will discuss other methods for determining these and other parameters of models.

3.2 Overfitting and Regularization

Directly minimizing squared-error can lead to an effect called **overfitting,** wherein we fit the training data extremely well (i.e., with low error), yet we obtain a model that produces very poor predictions on future test data whenever the test inputs differ from the training inputs (Figure 2(b)). Overfitting can be understood in many ways, all of which are variations on the same underlying pathology:

1. The problem is insufficiently constrained: for example, if we have ten measurements and ten model parameters, then we can often obtain a perfect fit to the data.

2. Fitting noise: overfitting can occur when the model is so powerful that it can fit the data and also the random noise in the data.

3. Discarding uncertainty: the posterior probability distribution of the unknowns is insufficiently peaked to pick a single estimate. (We will explain what this means in more detail later.)

There are two important solutions to the overfitting problem: adding prior knowledge and handling uncertainty. The latter one we will discuss later in the course.

In many cases, there is some sort of prior knowledge we can leverage. A very common assumption is that the underlying function is likely to be **smooth**, for example, having small derivatives. Smoothness distinguishes the examples in Figure 2. There is also a practical reason to prefer smoothness, in that assuming smoothness reduces model complexity: it is easier to estimate smooth models from small datasets. In the extreme, if we make no prior assumptions about the nature of the fit then it is impossible to learn and generalize at all; smoothness assumptions are one way of constraining the space of models so that we have any hope of learning from small datasets.

One way to add smoothness is to parameterize the model in a smooth way (e.g., making the width parameter for RBFs larger; using only low-order polynomial basis functions), but this limits the expressiveness of the model. In particular, when we have lots and lots of data, we would like the data to be able to "overrule" the smoothness assumptions. With large widths, it is impossible to get highly-curved models no matter what the data says.

Instead, we can add **regularization**: an extra term to the learning objective function that prefers smooth models. For example, for RBF regression with scalar outputs, and with many other types of basis functions or multi-dimensional outputs, this can be done with an objective function of the form:

$$E(\mathbf{w}) = \underbrace{||\mathbf{y} - \mathbf{B}\mathbf{w}||^2}_{\text{data term}} + \underbrace{\lambda||\mathbf{w}||^2}_{\text{smoothness term}} \tag{10}$$

This objective function has two terms. The first term, called the data term, measures the model fit to the training data. The second term, often called the smoothness term, penalizes non-smoothness (rapid changes in $f(x)$). This particular smoothness term ($||\mathbf{w}||$) is called **weight decay**, because it tends to make the weights smaller.[2] Weight decay implicitly leads to smoothness with RBF basis functions because the basis functions themselves are smooth, so rapid changes in the slope of f (i.e., high curvature) can only be created in RBFs by adding and subtracting basis functions with large weights. (Ideally, we might directly penalize smoothness, e.g., using an objective term that directly penalizes the integral of the squared curvature of $f(\mathbf{x})$, but this is usually impractical.)

[2]Estimation with this objective function is sometimes called Ridge Regression in Statistics.

This **regularized least-squares** objective function is still quadratic with respect to $\mathbf{w}$ and can be optimized in closed-form. To see this, we can rewrite it as follows:

$$\begin{aligned} E(\mathbf{w}) &= (\mathbf{y} - \mathbf{B}\mathbf{w})^T(\mathbf{y} - \mathbf{B}\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w} && (11)\\ &= \mathbf{w}^T\mathbf{B}^T\mathbf{B}\mathbf{w} - 2\mathbf{w}^T\mathbf{B}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w} + \mathbf{y}^T\mathbf{y} && (12)\\ &= \mathbf{w}^T(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I})\mathbf{w} - 2\mathbf{w}^T\mathbf{B}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} && (13) \end{aligned}$$

To minimize $E(\mathbf{w})$, as above, we solve the normal equations $\nabla E(\mathbf{w}) = \mathbf{0}$ (i.e., $\partial E/\partial w_i = 0$ for all i). This yields the following regularized LS estimate for $\mathbf{w}$:

$$\mathbf{w}^* = (\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I})^{-1}\mathbf{B}^T\mathbf{y} \quad (14)$$

3.3 Artificial Neural Networks

Another choice of basis function is the sigmoid function. "Sigmoid" literally means "s-shaped." The most common choice of sigmoid is:

$$g(a) = \frac{1}{1 + e^{-a}} \quad (15)$$

Sigmoids can be combined to create a model called an **Artificial Neural Network (ANN)**. For regression with multi-dimensional inputs $\mathbf{x} \in \mathbb{R}_2^K$, and multidimensional outputs $\mathbf{y} \in \mathbb{R}^{K_1}$:

$$\mathbf{y} = f(\mathbf{x}) = \sum_j \mathbf{w}_j^{(1)} g\left(\sum_k w_{k,j}^{(2)} x_k + b_j^{(2)}\right) + \mathbf{b}^{(1)} \quad (16)$$

This equation describes a process whereby a linear regressor with weights $\mathbf{w}^{(2)}$ is applied to $\mathbf{x}$. The output of this regressor is then put through the nonlinear Sigmoid function, the outputs of which act as features to another linear regressor. Thus, note that the *inner weights* $w^{(2)}$ are distinct parameters from the *outer weights* $\mathbf{w}_j^{(1)}$. As usual, it is easiest to interpret this model in the 1D case, i.e.,

$$y = f(x) = \sum_j w_j^{(1)} g\left(w_j^{(2)} x + b_j^{(2)}\right) + b^{(1)} \quad (17)$$

Figure 3(left) shows plots of $g(wx)$ for different values of w, and Figure 3(right) shows $g(x+b)$ for different values of b. As can be seen from the figures, the sigmoid function acts more or less like a step function for large values of w, and more like a linear ramp for small values of w. The bias b shifts the function left or right. Hence, the neural network is a linear combination of shifted (smoothed) step functions, linear ramps, and the bias term.

To learn an artificial neural network, we can again write a regularized squared-error objective function:

$$E(w, b) = ||\mathbf{y} - f(\mathbf{x})||^2 + \lambda||\mathbf{w}||^2 \quad (18)$$

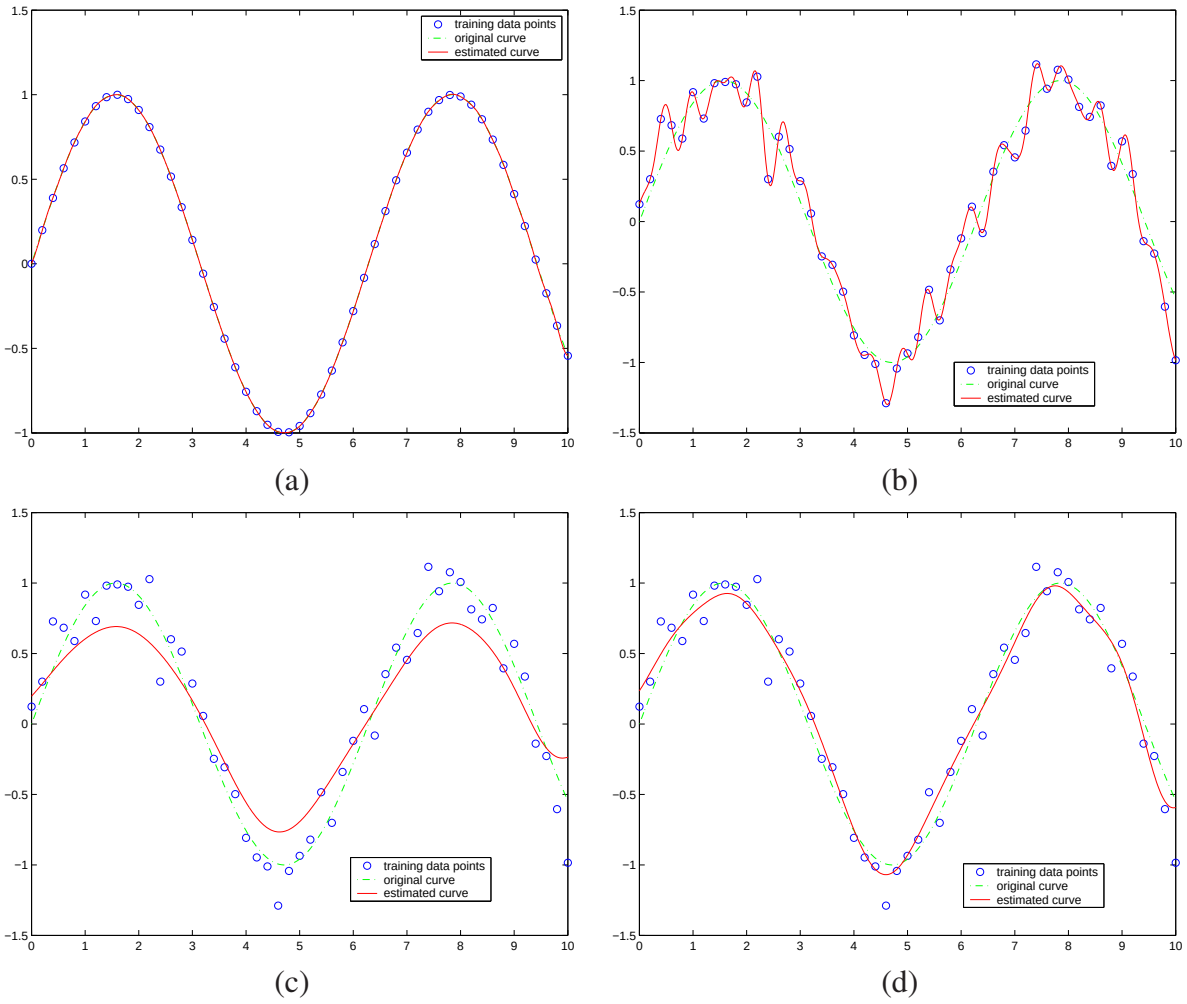

Figure 2: Least-squares curve fitting of an RBF. (a) Point data (blue circles) was taken from a sine curve, and a curve was fit to the points by a least-squares fit. The horizontal axis is x, the vertical axis is y, and the red curve is the estimated $f(\mathbf{x})$. In this case, the fit is essentially perfect. The curve representation is a sum of Gaussian basis functions. (b) **Overfitting**. Random noise was added to the data points, and the curve was fit again. The curve exactly fits the data points, which does not reproduce the original curve (a green, dashed line) very well. (c) **Underfitting**. Adding a smoothness term makes the resulting curve too smooth. (In this case, weight decay was used, along with reducing the number of basis functions). (d) Reducing the strength of the smoothness term yields a better fit.

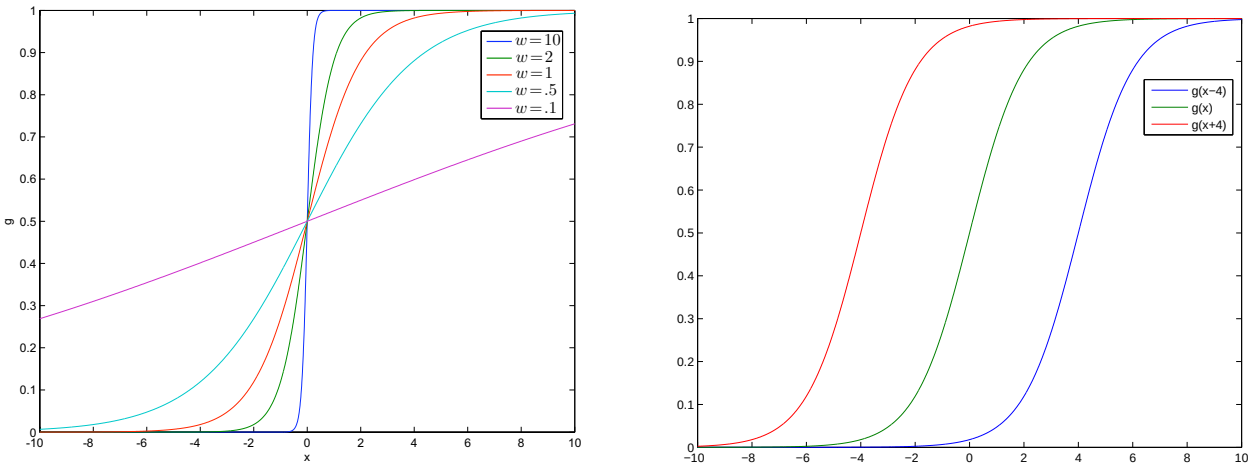

Figure 3: *Left:* Sigmoids $g(wx) = 1/(1+e^{-wx})$ for various values of w, ranging from linear ramps to smooth steps to nearly hard steps. *Right:* Sigmoids $g(x+b) = 1/(1+e^{-x-b})$ with different shifts b.

where $\mathbf{w}$ comprises the weights at both levels for all j. Note that we regularize by applying weight decay to the weights (both inner and outer), but not the biases, since only the weights affect the smoothness of the resulting function (why?).

Unfortuntely, this objective function cannot be optimized in closed-form, and numerical optimization procedures must be used. We will study one such method, gradient descent, in a later chapter.

3.4 K-Nearest Neighbors

At their heart, many learning procedures — especially when our prior knowledge is weak — amount to smoothing the training data. RBF fitting is an example of this. However, many of these fitting procedures require making a number of decisions, such as the locations of the basis functions, and can be sensitive to these choices. This raises the question: why not cut out the middleman, and smooth the data directly? This is the idea behind **K-Nearest Neighbors** regression.

The idea is simple. We first select a parameter K, which is the only parameter to the algorithm. Then, for a new input $\mathbf{x}$, we find the K nearest neighbors to $\mathbf{x}$ in the training set, based on their Euclidean distance $||\mathbf{x} - \mathbf{x}_i||^2$. Then, our new output $\mathbf{y}$ is simply an average of the training outputs for those nearest neigbors. This can be expressed as:

$$\mathbf{y} = \frac{1}{K} \sum_{i \in N_K(\mathbf{x})} \mathbf{y}_i \tag{19}$$

where the set $N_K(\mathbf{x})$ contains the indicies of the K training points closest to $\mathbf{x}$. Alternatively, we might take a weighted average of the K-nearest neighbors to give more influence to training points

close to $\mathbf{x}$ than to those further away:

$$\mathbf{y} = \frac{\sum_{i \in N_K(\mathbf{x})} w(\mathbf{x}_i)\mathbf{y}_i}{\sum_{i \in N_K(\mathbf{x})} w(\mathbf{x}_i)} \quad , \quad w(\mathbf{x}_i) = e^{-||\mathbf{x}_i - \mathbf{x}||^2/2\sigma^2} \tag{20}$$

where σ^2 is an additional parameter to the algorithm. The parameters K and σ control the degree of smoothing performed by the algorithm. In the extreme case of $K = 1$, the algorithm produces a piecewise-constant function.

K-nearest neighbors is simple and easy to implement; it doesn't require us to muck about at all with different choices of basis functions or regularizations. However, it doesn't compress the data at all: we have to keep around the entire training set in order to use it, which could be very expensive, and we must search the whole data set to make predictions. (The cost of searching can be mitigated with spatial data-structures designed for searching, such as k-d-trees and locality-sensitive hashing. We will not cover these methods here).

