3 Nonlinear Regression

Linear models are often insufficient to capture the real-world phenomena. That is, the relation between the inputs and the outputs we want to be able to predict are not linear. As a consequence, nonlinear models are often required. We are still interested in parameterized functions of the form

$$\mathbf{y} \;=\; f(\mathbf{x})\,. \tag{3.1}$$

In linear regression, we used $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, the parameters of which were $\mathbf{W}$ and $\mathbf{b}$. We estimate the parameters to fit the model to data.

In the case of nonlinear regression $f(\mathbf{x})$ is a nonlinear function. But what nonlinear function should we choose? In principle, $f(\mathbf{x})$ could be anything. It could involve linear functions, trigonometric functions, summations, and so on. However, the form we choose will have a major impact 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. That is, if we're modeling a linear process, then we'd want to use linear regression. If we were modeling a physical process, we could, in principle, model $f(\mathbf{x})$ using the appropriate equations from physics. And so on.

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

Also, 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 our estimated models to be smooth so that they do a better job in predicting values for previously unseen inputs. It is hoped that smoother models will tend to overfit the training data less and therefore generalize somewhat better, providing better predictions at new test inputs.

3.1 Basis Function Regression

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

$$y \;=\; f(x) \;=\; \sum_k w_k b_k(x) \tag{3.2}$$

for the case of 1D inputs x. 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

$$y \;=\; f(x) \;=\; \mathbf{b}(x)^T\mathbf{w} \tag{3.3}$$

[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" $b_k(x)$

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

$$b_0(x) = 1, \quad b_1(x) = x, \quad b_2(x) = x^2, \quad b_3(x) = x^3, \quad ... \tag{3.4}$$

With such a monomial basis, the regression model has the form

$$f(x) \;=\; \sum w_k \, x^k \;, \tag{3.5}$$

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

$$b_k(x) \;=\; \exp\left(-\frac{(x-c_k)^2}{2\sigma^2}\right) \;, \tag{3.6}$$

where $\exp a \equiv e^a$. Accordingly,

$$f(x) \;=\; \sum w_k \, b_k(x) \;=\; \sum w_k \exp\left(-\frac{(x-c_k)^2}{2\sigma^2}\right) \;, \tag{3.7}$$

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 (or estimated) from the training data.

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 the number of weights to be estimated is relatively small.

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

$$E(\mathbf{w}) \;=\; \sum_i (y_i - f(x_i))^2 \;=\; \sum_i \left(y_i - \sum_k w_k b_k(x) \right)^2 \tag{3.8}$$

To minimize this function with respect to $\mathbf{w}$, note that this objective function has the same form as the linear regression model in the previous chapter, except that the inputs are now the $b_k(x)$ values. In particular, E is still quadratic in the weight parameters, $\mathbf{w}$, and hence they can be estimated the same way. We therefore rewrite the objective function in matrix-vector form as follows,

$$E(\mathbf{w}) \;=\; ||\mathbf{y} - \mathbf{B}\mathbf{w}||^2 \;, \tag{3.9}$$

where $||\cdot||$ denotes the Euclidean (L2) norm, and the elements of the matrix $\mathbf{B}$ are given by $\mathbf{B}_{i,j} = b_j(x_i)$. That is, the element of $\mathbf{B}$ in row i, column j is $b_j(x_i)$. So when the model contains K basis functions, $\mathbf{B}$ is an $N \times K$ matrix with one row per data point.

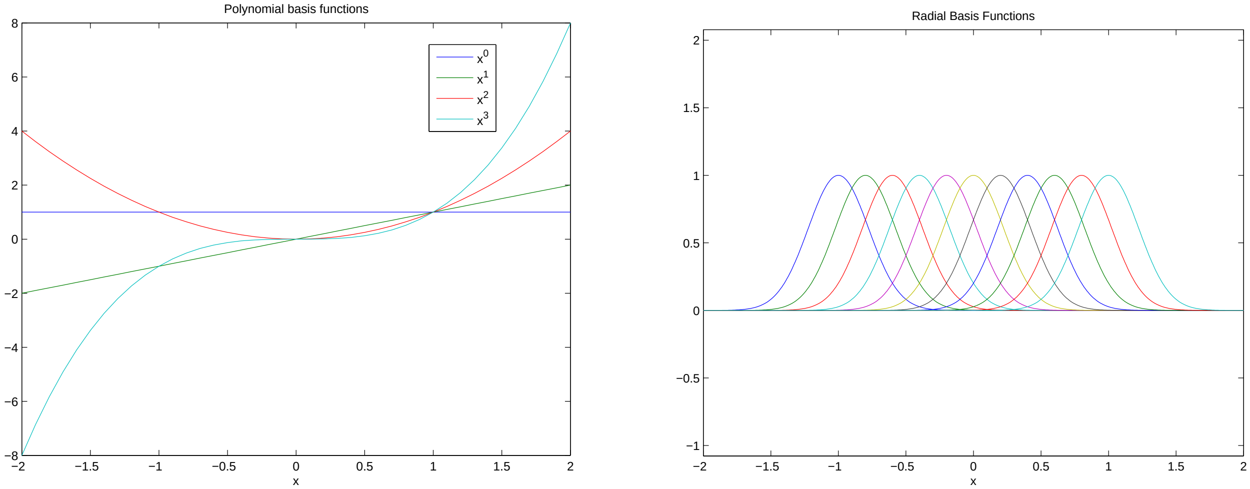


Figure 3.1: The first three basis functions of a polynomial basis, and Radial Basis Functions

Picking the other parameters. It turns out that finding optimal positions and widths of the RBF basis functions is not so simple, nor are the criteria we need to use to the select them. For example, if we were to optimize these parameters to minimize the squared-error, then we would end up with one basis function centered at each data point, and with tiny width. These RBF functions would exactly fit the data, but such models will rarely provide good predictions for inputs other than those in the training set. This is a problem.

The following heuristics 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 3.2(b)). Overfitting can be understood in many ways, all of which are variations on the same underlying pathology:

1. The problem is not sufficiently constrained: For example, if we have ten measurements and ten model parameters, then we can often obtain a perfect fit to the data. Usually you need much more data than there are model parameters.

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 terms of Bayesian model averaging later in the course.)

There are two important solutions to the overfitting problem, namely, 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 instance we might believe that, lacking any other information, our functions should have small derivatives. Smoothness distinguishes the examples in Figure 3.2. 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.

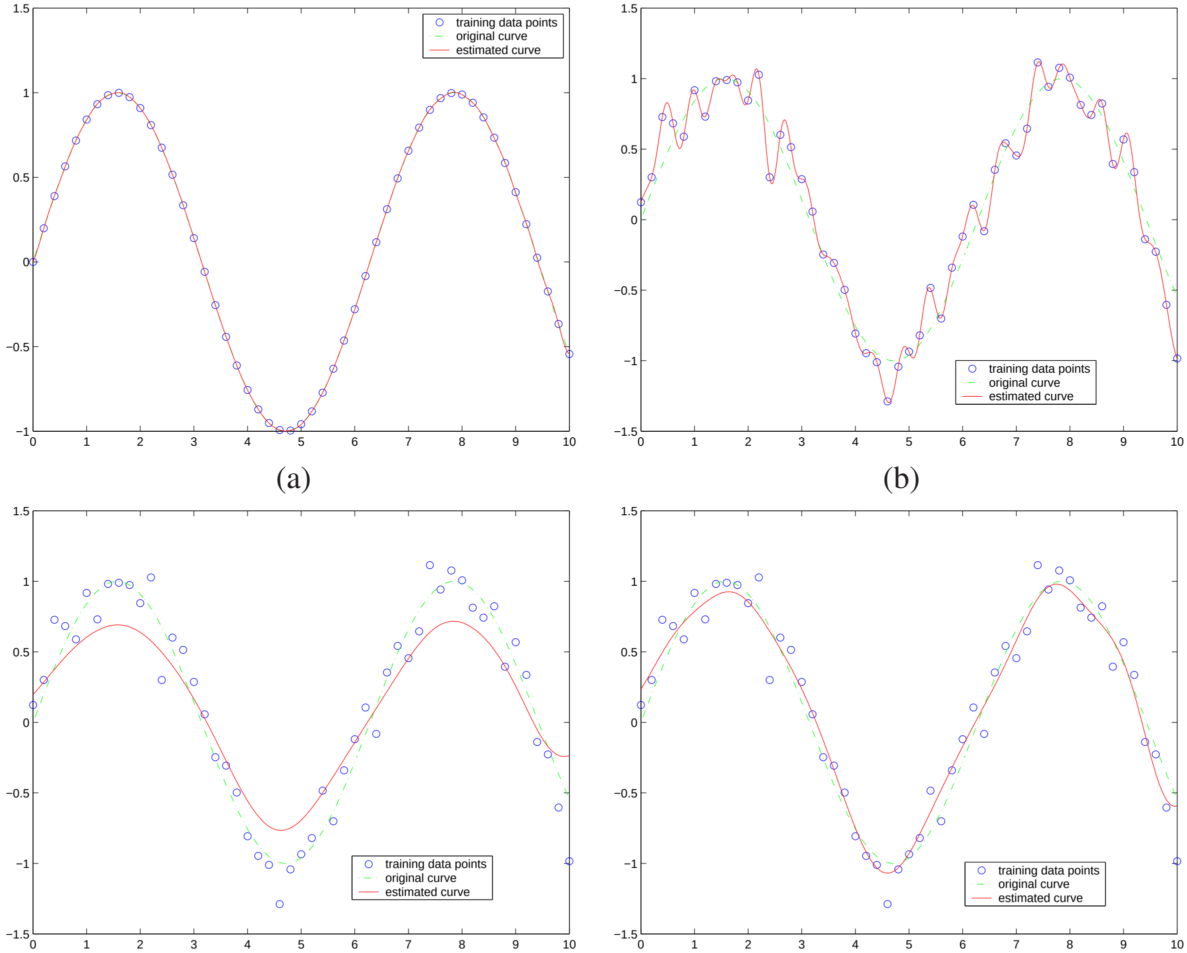
One way to add smoothness is to parameterize the model in a smooth way (e.g., making the width parameter for RBFs larger, using only low-order polynomial basis functions). But this limits the expressiveness of the model. In particular, when we have lots and lots of data, we would like the data to be 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**, in which a extra terms are added to the learning objective function, often to prefer (encourage) 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:

$$E(\mathbf{w}) \;=\; \underbrace{||\mathbf{y}-\mathbf{B}\mathbf{w}||^2}_{\text{data term}} + \underbrace{\lambda||\mathbf{w}||^2}_{\text{smoothness term}} \tag{3.10}$$

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

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



(a) (b)

(c) (d)

Figure 3.2: 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.

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.)

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:

$$\begin{aligned} E(\mathbf{w}) &= (\mathbf{y} - \mathbf{B}\mathbf{w})^T(\mathbf{y} - \mathbf{B}\mathbf{w}) + \lambda \mathbf{w}^T\mathbf{w} && (3.11) \\ &= \mathbf{w}^T\mathbf{B}^T\mathbf{B}\mathbf{w} - 2\mathbf{w}^T\mathbf{B}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w} + \mathbf{y}^T\mathbf{y} && (3.12) \\ &= \mathbf{w}^T(\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I})\mathbf{w} - 2\mathbf{w}^T\mathbf{B}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} && (3.13) \end{aligned}$$

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 $\mathbf{w}$:

$$\mathbf{w}^* = (\mathbf{B}^T\mathbf{B} + \lambda\mathbf{I})^{-1}\mathbf{B}^T\mathbf{y} \tag{3.14}$$

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:

$$g(a) = \frac{1}{1 + e^{-a}} \tag{3.15}$$

Among their many uses in machine learning, sigmoids are often used as the nonlinearity in **Artificial Neural Networks (ANN)**. For regression with multi-dimensional inputs $\mathbf{x} \in \mathbb{R}_2^K$, and multi-dimensional outputs $\mathbf{y} \in \mathbb{R}^{K_1}$, a sigmoid-based ANN has the form

$$\mathbf{y} = f(\mathbf{x}) = \sum_j \mathbf{w}_j^{(1)} g\left(\sum_k w_{k,j}^{(2)} x_k + b_j^{(2)}\right) + \mathbf{b}^{(1)} \tag{3.16}$$

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* $\mathbf{w}_j^{(1)}$. It is easiest to interpret this model in the 1D case, i.e.,

$$y = f(x) = \sum_j w_j^{(1)} g\left(w_j^{(2)} x + b_j^{(2)}\right) + b^{(1)} \tag{3.17}$$

Figure 3.3(left) shows plots of $g(wx)$ for different values of w, and Figure 3.3(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, roughly speaking, a linear combination of shifted (smoothed) step functions, linear ramps, and the bias term.

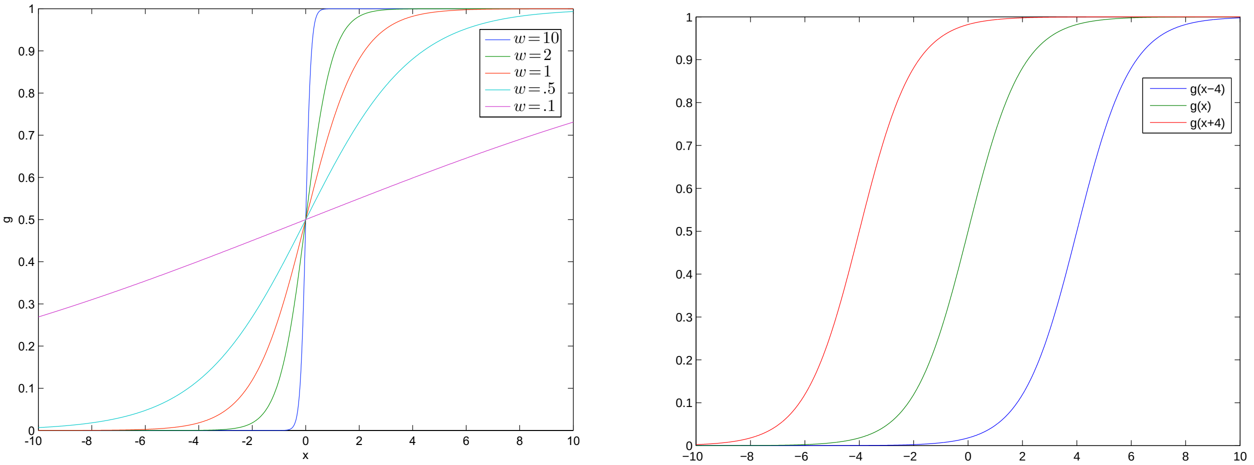


Figure 3.3: *Left:* Sigmoids $g(wx) = 1/(1 + e^{-wx})$ for various values of w, ranging from linear ramps to smooth steps to nearly hard steps. *Right:* Sigmoids $g(x + b) = 1/(1 + e^{-x-b})$ with different shifts b.

Learning an ANN entails estimating its parameters, namely the network weights etc. To this end we can again write a regularized squared-error objective function:

$$E(w, b) \;=\; ||\mathbf{y} - f(\mathbf{x})||^2 + \lambda||\mathbf{w}||^2 \tag{3.18}$$

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, so numerical optimization procedures must be used. We will study one such method, gradient descent, in a later chapter.

3.4 k-Nearest Neighbors

In essence, many learning procedures — especially when our prior knowledge is weak — amount to smoothing the training data. RBF fitting is one 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 very 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 to the input, i.e., $||\mathbf{x} - \mathbf{x}_i||^2$. Then, our prediction $\mathbf{y}$ is simply an average of the training outputs in the set of k nearest neigbors. This can be expressed as:

$$\mathbf{y} \;=\; \frac{1}{k} \sum_{i \in N_k(\mathbf{x})} \mathbf{y}_i \;, \tag{3.19}$$

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 closer to $\mathbf{x}$ than to those further away; e.g.,

$$\mathbf{y} \;=\; \frac{\sum_{i \in N_k(\mathbf{x})} w(\mathbf{x}_i)\mathbf{y}_i}{\sum_{i \in N_k(\mathbf{x})} w(\mathbf{x}_i)} \;\;, \quad w(\mathbf{x}_i) \;=\; e^{-||\mathbf{x}_i - \mathbf{x}||^2/2\sigma^2} \;, \tag{3.20}$$

where σ^2 is an additional parameter that we need to specify. 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. That is, we have to keep around the entire training set in order to use it, which is very expensive, and we must search the entire dataset to make predictions. (The cost of searching can be mitigated with spatial data-structures designed for searching, such as KD-trees and locality-sensitive hashing. We will not cover these methods here).