An Introduction to Probabilistic Graphical Models

Michael I. Jordan
University of California, Berkeley

June 30, 2003

Chapter 6

Linear Regression and the LMS algorithm

In the following chapters we discuss elementary building blocks for graphical models. We begin with the simple case of a single continuous-valued node whose mean is a linear function of the values of its parents. The parents can be discrete or continuous.

In specifying the linear regression model in Chapter 5, we made several assumptions in addition to the linearity assumption, in particular the assumption of IID sampling and the assumption of a Gaussian distribution for the variation around the conditional mean. These latter assumptions yielded a fully specified probabilistic model, enabling us to define a likelihood and thereby invoke frequentist or Bayesian statistical methods to estimate parameters. It might be useful, however, to step momentarily outside of the probabilistic framework and ask why we consider a parameter estimation problem to be well posed once we have defined a "fully specified probabilistic model." In the current chapter, we address this foundational issue in a rather concrete way, taking advantage of the simplicity of the linear model to bring to the fore a different set of intuitions about parameter estimation. We begin by making the linearity assumption, but then let geometric rather than probabilistic intuitions be our guide. In particular, we view each data point as imposing a linear constraint on the parameters and treat parameter estimation as a (deterministic) constraint satisfaction problem. We focus on obtaining algorithms that solve this constraint satisfaction problem, exploiting the geometric framework to analyze the convergence of these algorithms.

The emphasis on constraint satisfaction algorithms in the current chapter has the advantage of focusing attention on some computational issues that are important in practice and were glossed over in our purely statistical discussion in Chapter 5. In particular, we will introduce the distinction between "batch" and "on-line" algorithms, a distinction which is of importance in real-time applications of statistical modeling and in situations involving large data sets.

At the end of the chapter, we return to the probabilistic perspective, showing that there is a natural correspondence between the (Euclidean) geometry underlying the constraint satisfaction formulation and the statistical assumptions alluded to above. Thus, we can view the excursion into geometry as providing support for the statistical perspective in Chapter 5; thus encouraged, we will be less bashful about bringing probabilistic machinery to bear at the outset in future chapters. At the same time, we will continue to seek external support for probabilistic assumptions, particularly when they shed light on computational concerns.

6.1 Batch and on-line algorithms

Let us consider in some more detail how data points may be presented to the learner. We wish to distinguish two basic situations—the setting of "batch" presentation, in which data are available as a block, and the "on-line" setting in which data arrive sequentially. Both settings arise naturally in practice: In many problems it is necessary to respond in real time, and on-line methods are dictated; in other situations our only interest is in a final answer—the best answer that we can obtain given a certain data-gathering budget—and in such cases batch methods are natural.

On the other hand, we are often free to take either the batch or the on-line point of view on a learning problem—a sequential data stream can be stored for subsequent analysis as a block, and a block of data can be accessed sequentially. Moreover, a theoretical understanding of algorithms for parameter estimation is enhanced by approaching the problem from both points of view. We will see that the on-line point of view yields simple, intuitive algorithms, but a full analytical understand of on-line algorithms can be difficult, and we therefore turn to a related batch analysis to enhance understanding. On the other hand, batch methods are often usefully understood by taking an on-line point of view—in particular, large-scale batch problems generally require iterative algorithms that sweep repeatedly through the data. These sweeps can often be usefully analyzed as on-line algorithms.

A great deal of insight can be obtained by considering the elemental problem of updating the parameters of a linear model based on the presentation of a single data point. Let us begin with a discussion of the geometry underlying this problem, and show how simple geometric intuition leads us to an on-line algorithm known as the *LMS algorithm*. The acronym "LMS" refers to "least mean squares," which, as we shall see, reflects the fact that the algorithm can be viewed as an optimization or constraint satisfaction procedure.

6.2 The LMS algorithm

Let us begin with a minimum of probabilistic pretension and consider the core of the linear model—the linear dependence of one variable on another. We consider the following question: Suppose that we have a pair of observed variables x_n and y_n that we assume are related linearly. What should a learning algorithm do when presented with a data point consisting of the pair (x_n, y_n)? We shall be very naive and see if we can get any clues as to how to design a learning algorithm by considering the vector space geometry that characterizes the model.

We wish to express y_n as a linear function of x_n:

$$y_n = \theta^T x_n + \epsilon_n, \tag{6.1}$$

where θ is a parameter vector. Let us view ϵ_n as a deterministic "error term" whose presence in Eq. (6.1) is an admission that we don't necessarily expect to be able to express y_n perfectly as a

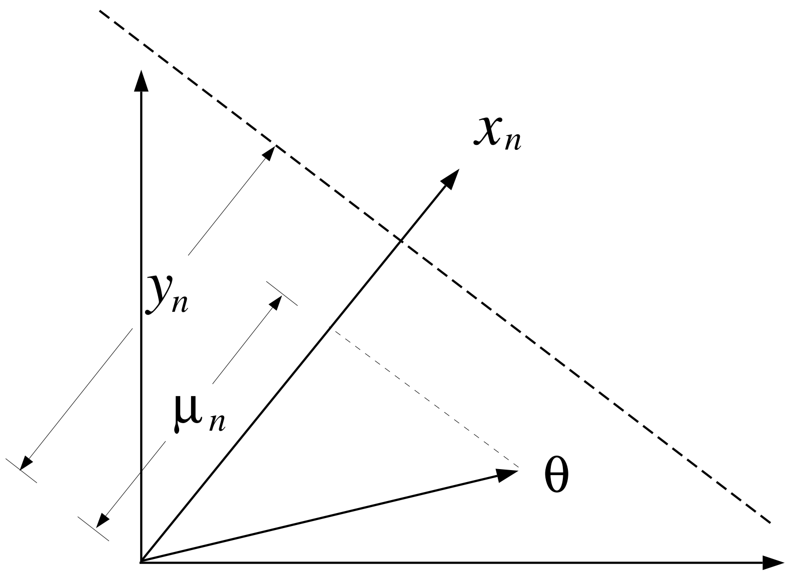

Figure 6.1: The geometry associated with the LMS algorithm. The figure shows the projection μ_n of the parameter vector θ on the input vector x_n. Also shown is the output value y_n as a distance along the x_n vector. The dashed line is the set of all vectors θ that have a projection of y_n and thus are solutions. The error associated with θ is the distance $(y_n - \mu_n)$. Changing θ by the vector $(y_n - \mu_n)x_n$ thus yields a solution vector.

linear function of x_n. In particular, let us forgo endowing ϵ_n or any of the other terms in Eq. (6.1) with probability distributions.

Figure 6.1 displays the vectors x_n and θ as well as the projection of θ on x_n, whose value we denote by μ_n. The projection μ_n is the inner product $\theta^T x_n$ divided by the norm of x_n.[1] Let us suppose for simplicity (temporarily) that x_n has norm one, so that the inner product $\theta^T x_n$ is the same as the projection μ_n. Now consider the problem of finding a vector θ that maps x_n to y_n exactly; that is, a vector such that ϵ_n is zero. Clearly we require a vector θ whose projection onto x_n is equal to y_n, and as the figure shows, there is a line of possible solutions that is orthogonal to x_n. Any vector along this line projects to the desired value y_n. We can view the data point (x_n, y_n) as imposing a constraint upon the vector θ that it lie along this line.

How might we design a learning algorithm to update the current value of the parameter vector

[1]Recall the fundamental relationship:

$$\cos\alpha = \frac{\theta^T x_n}{\|\theta\|\|x_n\|}, \tag{6.2}$$

where α is the angle between θ and x_n. From this relationship we obtain

$$\mu_n = \frac{\theta^T x_n}{\|x_n\|}. \tag{6.3}$$

for the projection $\mu_n = \|\theta\| \cos\alpha$.

θ such that the new value of θ meets the constraint and lies on the solution line? Although there are an infinity of possible directions that we could move, note that there are two *natural* directions available to us: the direction associated with x_n and that associated with θ. Let us be very naive and decide that we should choose one of these two directions as the direction in which to update θ.

Although it is possible to figure out how far to move along the θ direction so as to intersect the solution line, it is rather easier to figure out how far to move along the x_n direction, given the orthogonality of x_n and the solution line. Let us opt for simplicity and choose to follow x_n.

Given that x_n is a unit vector, and given that the error we incur using the current parameter vector is the difference $(y_n - \theta^T x_n)$, it is clear that we should move the parameter vector a distance $(y_n - \theta^T x_n)$ in the x_n direction. Thus:

$$\theta^{(t+1)} = \theta^{(t)} + (y_n - \theta^{(t)T} x_n) x_n, \tag{6.4}$$

where $\theta^{(t)}$ is the estimated value of θ at the tth step of the algorithm. This algorithm jumps to the solution line.

If the vector x_n is not a unit vector, then we need to scale all of our distances by the norm of x_n. The projection is now $\theta^T x_n / \|x_n\|$ and thus we need to choose a parameter vector whose projection onto x_n is $y_n / \|x_n\|$. This implies that the error we incur using θ is given by $(y_n - \theta^T x_n)/\|x_n\|$, which is the amount we need to move in the direction of x_n. The unit vector in this direction is given by $x_n / \|x_n\|$; thus we obtain the following learning algorithm:

$$\theta^{(t+1)} = \theta^{(t)} + \frac{1}{\|x_n\|^2}(y_n - \theta^{(t)T} x_n) x_n, \tag{6.5}$$

which again hops to the solution line in a single step.

More generally, we express our learning algorithm in the following form:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(y_n - \theta^{(t)T} x_n) x_n, \tag{6.6}$$

where ρ is a free parameter known as the "step size." Our analysis has shown that the choice $\rho = 1/\|x_n\|^2$ yields an algorithm that hops to the solution line in a single step. It is also easy to see that if $0 < \rho < 2/\|x_n\|^2$, then on repeated presentations of x_n the algorithm will converge to the solution line asymptotically.

The algorithm in Equation 6.6 is the LMS algorithm.

6.2.1 Multiple data points

Let us now consider the case in which multiple data points are available. In particular we suppose that we have a "training set" $\mathcal{X} = \{(x_n, y_n)\}_{n=1}^N$, where N, the number of data points, is at least as large as k, the dimensionality of the parameter vector.

Let us begin by considering the simplest case, in which $N = k$; let us also assume for simplicity that the vectors x_n are linearly independent. Under these conditions, the model given by Equation 6.1 imposes a set of k linearly independent equations on k unknowns. This implies the existence of a unique parameter vector θ that achieves $\epsilon_n = 0$ for each n. Will the LMS algorithm find this solution?

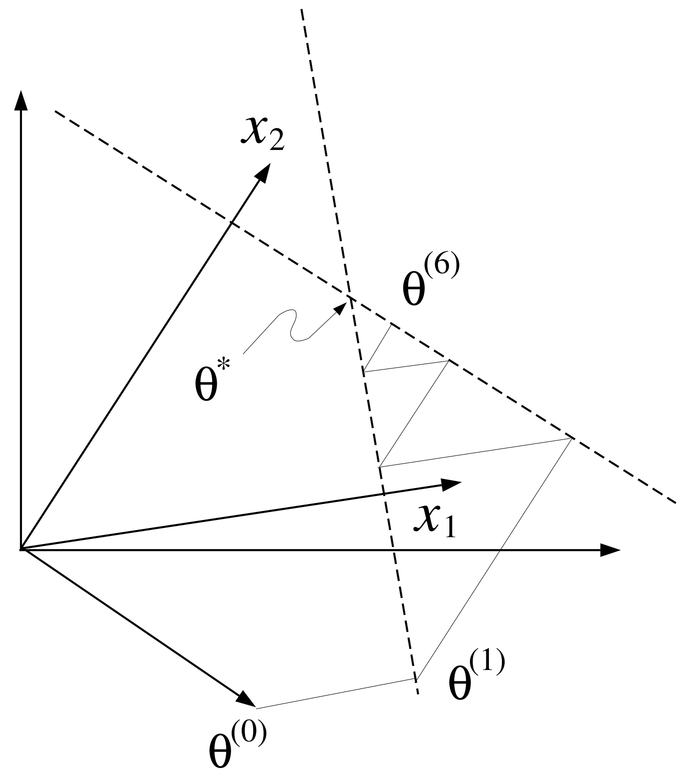

Figure 6.2: The geometry associated with the LMS algorithm in the case of two input vectors x_1 and x_2. Associated with each vector is a line of solutions and the intersection of these lines is the vector θ^* that solves the problem for both vectors. We show the path taken by the LMS algorithm upon repeated presentations of x_1 and x_2.

Figure 6.2 presents an example for the case of $N = 2$. As shown in the figure, each of the directions determined by the vectors x_1 and x_2 is associated with a solution line of vectors θ that map the given x_i to the corresponding y_i. The value θ^* that maps both vectors x_n to their desired values lies at the intersection of these two lines. Assuming that the training regime alternates between the two data points, we see that the LMS algorithm takes a zigzag path, following first the x_1 direction and then the x_2 direction. It seems clear, and it is in fact true (as we will show), that there exists a maximum value of the step size for which the algorithm converges to the solution.

If we turn to the case of $N > k$, we expect to see a qualititatively similar behavior in which LMS takes a zigzag path through the parameter space. In this case, however, we have an overdetermined set of equations and the lines that achieve ϵ_n for the various data points do not meet at a single point (see Figure 6.3). Given that LMS always moves from the current θ towards the line of solutions corresponding to the current data point, we see that we cannot expect the algorithm to move to a single point and stay put. We do expect, however, that the LMS algorithm should "converge" towards a small region of the parameter space, given an appropriate choice of the step size. Making further progress on these issues requires us to characterize more formally the constraint satisfaction

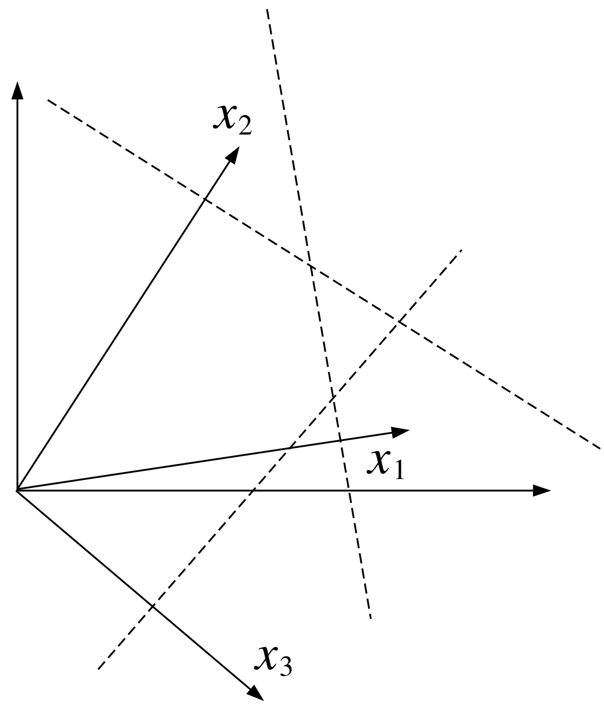

Figure 6.3: The geometry associated with the LMS algorithm in the case of three input vectors x_1, x_2 and x_3. Associated with each vector is a line of solutions. In general these lines do not intersect.

problem underlying the algorithm.

6.3 The sum of squares cost function and the normal equations

The approach that we pursue—which dates back to Gauss if not before—is to search for parameter vectors θ that yield "small" values of ϵ_n. We need to characterize what we mean by "small," and decide how to combine the errors for different values of n. To make these decisions we again reason geometrically. We now work in a different geometry, however, namely an N-dimensional vector space, where N is the number of data points.

Let y denote a column vector with components y_n and let $\hat{y}$ denote a column vector with components $\hat{y}_n = \theta^T x_n$. These are vectors in an N-dimensional vector space. We want to express the relationship between these vectors in a way that reveals more of the geometry behind the linear model. To do so, let X represent the matrix whose nth row is the row vector x_n^T. We write:

$$\hat{y} = X\theta, \tag{6.7}$$

showing that $\hat{y}$ lies in the column space of the matrix X. Each column of X corresponds to a particular component of the vector x_n, and the set of columns of X can be viewed as spanning a vector subspace (see Figure 6.4). The vector $\hat{y}$ lies in this vector subspace. The vector y, on the

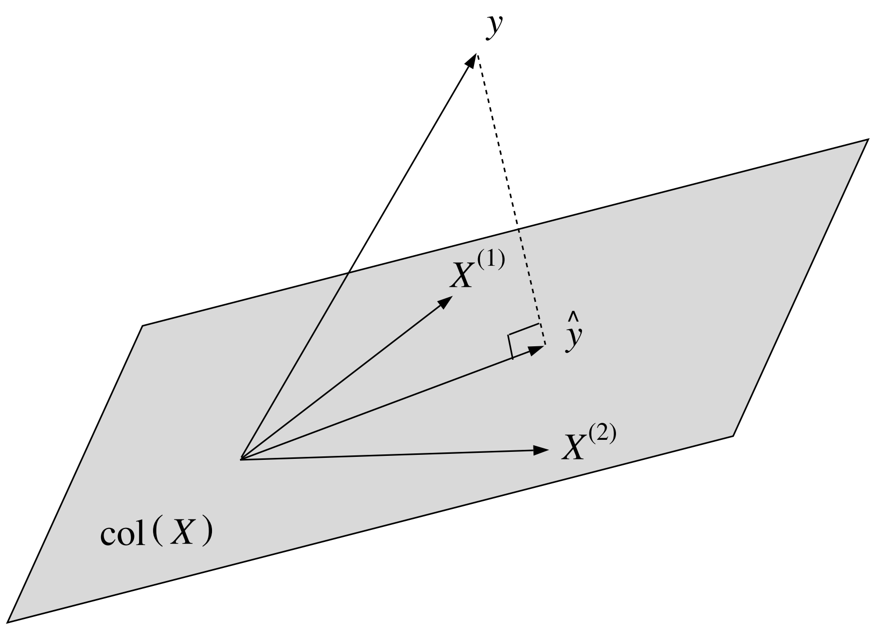

Figure 6.4: A geometric perspective on the linear regression problem for the case of the matrix X having two columns. Let $X^{(1)}$ represent the first column of X and let $X^{(2)}$ represent the second column. Denote the column space of X as $\mathrm{col}(X)$. The approximating vector $\hat{y}$ lies in this vector subspace, while the data vector y generally lies outside of this subspace. We wish to find a vector $\hat{y}$ that is the orthogonal projection of y on $\mathrm{col}(X)$.

other hand, generally lies outside of this vector subspace, reflecting the fact that in general the errors ϵ_n cannot simultaneously be zero.

Our problem reduces to choosing a vector $\hat{y}$ in a vector subspace that best represents a vector y outside of the subspace. A natural solution, from a geometric point of view, is to choose the *orthogonal projection* of y onto the subspace. We will solve the problem of finding a vector θ^* that yields this projection in three different ways.

Our first solution appeals directly to the geometry in Figure 6.4. In particular, for $\hat{y}$ to be the orthogonal projection of y on the column space of X, the difference vector $\epsilon = y - \hat{y}$ must be orthogonal to this vector subspace. Thus $y - \hat{y} = y - X\theta^*$ must be orthogonal to the columns of X, or, equivalently, orthogonal to the rows of X^T. This yields:

$$X^T(y - X\theta^*) = 0 \tag{6.8}$$

which implies

$$X^T X\theta^* = X^T y. \tag{6.9}$$

These equations, which characterize an optimizing vector θ^*, are referred to as the *normal equations.*

There is an equivalent characterization of the orthogonal projection in terms of minimal Euclidean length; this characterization leads us to a calculus-based derivation of the normal equations. In particular, let us choose $\hat{y}$ such that the error vector $\epsilon = y - \hat{y}$ has minimal Euclidean length. Thus, working (equivalently) with the squared length, we wish to minimize the *least squares cost function* $J(\theta)$:

$$J(\theta) \triangleq \frac{1}{2}\sum_{n=1}^{N}\epsilon_n^2 = \frac{1}{2}\sum_{n=1}^{N}(y_n - \theta^T x_n)^2, \tag{6.10}$$

with respect to θ.[2]

Differentiating J with respect to the ith component, θ^i, of the vector θ, we obtain:

$$\frac{\partial J}{\partial \theta^i} = -\sum_{n=1}^{N}(y_n - \theta^T x_n)x_n^i, \tag{6.11}$$

where x_n^i is the ith component of the vector x_n. Collecting these partial derivatives into a vector we obtain the following gradient:

$$\nabla_\theta J = -\sum_{n=1}^{N}(y_n - \theta^T x_n)x_n, \tag{6.12}$$

which we must set to zero to obtain conditions on the optimizing solution θ^*.

To obtain an explicit solution it is useful to make use of the matrix X and write the gradient as a single matrix equation. Recalling that X has the vectors x_n on its rows, we can view Eq. (6.12) as a sum of the rows of X, weighted by the values $(y_n - \theta^T x_n)$. Equivalently this sum is the sum

[2]The factor of 1/2 is included for convenience; it cancels the factor arising from the exponent of 2 when we take derivatives.

of the columns of X^T. Recalling that the values $\theta^T x_n$ are the components of the vector $\hat{y} = X\theta$, we have:

$$\nabla_\theta J = -X^T(y - X\theta). \tag{6.13}$$

Finally, setting to zero we obtain:

$$X^T(y - X\theta^*) = 0, \tag{6.14}$$

or equivalently:

$$X^T X\theta^* = X^T y, \tag{6.15}$$

which are the normal equations.

In Appendix XXX, we provide a short review of matrix and vector derivatives, which allows the reader to go directly from the cost function expressed in vector notation as:

$$\begin{aligned} J(\theta) &= \frac{1}{2}(y - X\theta)^T(y - X\theta) & (6.16) \\ &= \frac{1}{2}(y^T y - 2y^T X\theta + \theta^T X^T X\theta), & (6.17) \end{aligned}$$

directly to the gradient:

$$\nabla_\theta J = -X^T(y - X\theta), \tag{6.18}$$

from which we again obtain obtain the normal equations by setting to zero.

In most situations of practical interest, the number of data points N is larger than the dimensionality k of the input space and the matrix X has full column rank. If this condition holds, then it is easy to verify that $X^T X$ is necessarily invertible and thus we can express θ^* explicitly as follows:

$$\theta^* = (X^T X)^{-1} X^T y. \tag{6.19}$$

Moreover, if we take a second derivatives of J with respect to θ we find that the Hessian matrix of J is given by $X^T X$ (see Appendix XXX). The assumption that $X^T X$ is invertible implies that $X^T X$ is positive definite, and thus the critical point that we have found is a minimum. The solution to the normal equations provides the unique solution to the constraint satisfaction problem.

In Section **??** we discuss the case in which X has less than full column rank, and develop a *regularization* method to handle this case.

In the setting of "batch" presentation of data, in which data are available as a block, we can form the matrix X and the vector y and solve the normal equations. There are two major classes of methods for solving these equations: *direct methods* and *iterative methods.* The former class of methods, of which Gaussian elimination and QR decomposition are classical examples, converge in a finite number of steps. Iterative methods, which converge in a limiting sense, are of interest in the setting of particularly large problems, where direct approaches can be infeasible computationally.

Our next task is to try to understand the link between the two geometries that we have studied—the k-dimensional geometry of Figure 6.1 and the N-dimensional geometry of Figure 6.4. We also want to understand the relationship between the normal equations and the LMS algorithm. In the following section, we forge these links via the derivation of a steepest descent algorithm for solving the normal equations. This algorithm can be viewed as an example—one of many—of an iterative method for the batch case. Our goal, however, is not to explore iterative solution methods for the batch case (indeed there are more sophisticated methods than steepest descent). Rather, we wish to use the normal equations and their solution via steepest descent as a point of departure for understanding the on-line case.

6.4 Steepest descent and the LMS algorithm

Following the negative of the gradient in Eq. (6.12) we obtain the following *steepest descent* algorithm:

$$\theta^{(t+1)} = \theta^{(t)} + \rho \sum_{n=1}^{N} (y_n - \theta^{(t)T} x_n) x_n, \tag{6.20}$$

where $\theta^{(t)}$ is the parameter vector at the tth step of the iteration and where ρ is the step size. The algorithm is initialized at an arbitrary vector $\theta^{(0)}$ and iterates until a convergence criterion is met.

The steepest descent algorithm involves a sum over all N input vectors, thus the algorithm is a batch algorithm. Note that this aspect of the algorithm can render it rather inefficient, and this inefficiency motivates us to consider on-line approaches. In particular, if N is large, say in the millions, then the algorithm can spend an inordinate amount of time passing through the training set in order to compute the gradient, at which point it takes a step in the parameter space. Given that one of the motivations for studying iterative algorithms is to be able to handle very large problems, this feature of steepest descent is disconcerting. Note moreover that if the data set is redundant—a common occurrence with large data sets—then it might not be necessary to sum all of the N terms in Eq. (6.20) to obtain an accurate estimate of the direction of the gradient (the magnitude of the gradient is irrelevant because it is being scaled by a constant ρ that is under our control). In such situations, algorithms that take a sum over a subset of the data—a "mini-batch"—can often be significantly more efficient than the full batch algorithm. Indeed, in the limiting case we can view a single term, $-(y_n - \theta^T x_n)x_n$, as providing a rough estimate of the direction of the gradient. It may be advantageous to go ahead and follow this rough estimate and make progress in the parameter space rather than waiting to obtain a better estimate. This logic leads to the following algorithm, which adjusts the parameter vector according to the estimated gradient based on a single data point:

$$\theta^{(t+1)} = \theta^{(t)} + \rho (y_n - \theta^{(t)T} x_n) x_n. \tag{6.21}$$

This is of course the LMS algorithm. We see that the LMS algorithm can be viewed as an approximation to the steepest descent algorithm, where the approximation involves replacing the sum obtained in the batch algorithm with a single term. Such an approximation is referred to as a "stochastic gradient" algorithm, where "stochastic" refers to an assumption that the choice of data point (x_n, y_n) is made according to a stochastic process.

Let us emphasize that although LMS can be viewed as an approximation to steepest descent, it is often a much superior algorithm. Because it requires significantly less work per parameter update, it can converge significantly faster than steepest descent.

We are now in a position to learn something more about the convergence of the LMS algorithm. From the normal equations we have a characterization of the vector toward which we expect the LMS algorithm to tend, and from the steepest descent equations we have the possibility of characterizing the path that LMS will be expected to follow on average (under an appropriate stochastic analysis). In particular we may hope to learn something about the maximum possible value of ρ.

We present two analyses—one algebraic and one geometric—that yield the sought-after results. Both analyses involve analyzing the shape of the quadratic cost function J in the neigborhood of its minimum.

6.4.1 An algebraic convergence analysis[3]

One way to understand the convergence of the steepest descent algorithm in Eq. (6.20) is to unfold the recursion and solve the resulting equation.

In particular, letting $\theta^{(t)}$ represent the parameter vector at the tth iteration of the algorithm, we have:

$$
\begin{aligned}
\theta^{(t+1)} &= \theta^{(t)} + \rho \sum_{n=1}^{N} (y_n - \theta^{(t)T} x_n) x_n && (6.22)\\
&= \theta^{(t)} + \rho \sum_{n=1}^{N} x_n y_n - \rho \sum_{n=1}^{N} (x_n x_n^T) \theta^{(t)} && (6.23)\\
&= \theta^{(t)} + \rho X^T y - \rho X^T X \theta^{(t)} && (6.24)\\
&= (I - \rho X^T X)\theta^{(t)} + \rho X^T y. && (6.25)
\end{aligned}
$$

Expanding the recursion, we have:

$$
\begin{aligned}
\theta^{(t+1)} &= (I - \rho X^T X)\theta^{(t)} + \rho X^T y && (6.26)\\
&= (I - \rho X^T X)\left[(I - \rho X^T X)\theta^{(t-1)} + \rho X^T y\right] + \rho X^T y && (6.27)\\
&= (I - \rho X^T X)^{t+1}\theta^{(0)} + \rho \sum_{i=0}^{t} (I - \rho X^T X)^i X^T y. && (6.28)
\end{aligned}
$$

We now let t go to infinity. Let us assume for now that the first term goes to zero as t goes to infinity—we will then return to this term and derive a condition that ensures that it goes to zero. Thus we have:

$$
\begin{aligned}
\theta^{(\infty)} &= \rho \sum_{i=0}^{\infty} (I - \rho X^T X)^i X^T y && (6.29)\\
&= \rho(\rho X^T X)^{-1} X^T y && (6.30)\\
&= (X^T X)^{-1} X^T y, && (6.31)
\end{aligned}
$$

[3]The material in this section is optional; it will not be needed in later chapters.

which are the normal equations. We have thus shown that the steepest descent algorithm converges to the minimum of the cost function, under the assumption that the first term in Eq. (6.28) converges to zero.

Let us now consider the matrix power $(I - \rho X^T X)^{t+1}$ as $t \to \infty$. In general, to show that a matrix power converges to zero we need to show that its largest eigenvalue is less than one in absolute value. Now it is easy to verify that if λ is an eigenvalue of $(I - B)$ for a matrix B, then $1 - \lambda$ is an eigenvalue of B. Thus the absolute values of the eigenvalues of $(I - B)$ are less than one if any only if the absolute values of the eigenvalues of B are between zero and two. Thus we have the condition:

$$0 < \lambda_{\max}[\rho X^T X] < 2, \tag{6.32}$$

where $\lambda_{\max}$ represents the maximum eigenvalue of a matrix, or equivalently:

$$0 < \rho < 2/\lambda_{\max}[X^T X]. \tag{6.33}$$

This is the condition for convergence; the step size ρ can be no larger than two divided by the maximum eigenvalue of $X^T X$.

6.4.2 A geometric convergence analysis[4]

To get a better understanding of the convergence condition that we have just derived, let us rederive it from a geometric point of view.

Our cost function is a quadratic function in the components θ^i and can be plotted as a set of elliptical contours in the parameter space. In particular, for the example shown earlier in Figure 6.2, the corresponding contours are shown in Figure 6.5. Let us take a moment to understand how to obtain these contours.

We know that the minimum of the cost function is achieved by the vector θ^* that solves the normal equations. Our analysis will be simplified if we choose this optimizing point as the origin of our coordinate system. We choose new coordinates $\phi = \theta - \theta^*$ and express the cost function in these new coordinates:

$$\begin{aligned}
J(\phi) &= \frac{1}{2}\sum_{n=1}^{N}(y_n - \theta^T x_n)^2 \\
&= \frac{1}{2}(y - X\theta)^T(y - X\theta) \\
&= \frac{1}{2}(y - X(\phi + \theta^*))^T(y - X(\phi + \theta^*)) \\
&= \frac{1}{2}(y^T y - \theta^{*T} X^T y + \phi^T X^T X \phi),
\end{aligned}$$

where in passing from the third line to the fourth line we have expanded the quadratic expression and used the fact that θ^* solves the normal equations. In the new coordinates we see that the cost function is expressed simply as:

$$J(\phi) = C + \frac{1}{2}\phi^T X^T X \phi, \tag{6.34}$$

[4]The material in this section is optional; it will not be needed in later chapters.

Figure 6.5: The contours of the cost function $J(\theta)$ for the example in Figure 6.2.

where $C = y^T y - \theta^{*T} X^T y$ is a constant.

We now rotate the coordinate system so that the axes point along the major and minor axes of the ellipse. This is achieved by making use of the eigenvectors of $X^T X$. In particular, let A be the matrix whose column vectors are the eigenvectors of $X^T X$. We have:

$$X^T X = A \Lambda A^T, \tag{6.35}$$

where Λ is a diagonal matrix whose elements are the eigenvalues λ_i of $X^T X$. Note also that the fact that $X^T X$ is a symmetric matrix implies that A is orthogonal. Thus we have:

$$A^T X^T X A = \Lambda. \tag{6.36}$$

Now choose new coordinates $\psi = A^T \phi$. We obtain:

$$J(\psi) = C + \frac{1}{2}(A\psi)^T X^T X (A\psi) \tag{6.37}$$

$$= C + \frac{1}{2}\psi^T A^T X^T X A \psi \tag{6.38}$$

$$= C + \frac{1}{2}\psi^T \Lambda \psi. \tag{6.39}$$

This final equation is simply the weighted sum of squares of the components of ψ, with weights given by the eigenvalues λ_i. Setting $J(\psi)$ equal to a constant yields the equation of an ellipsoid.

Let us now express the steepest descent equation in the new coordinates. We write the equation in matrix notation (cf. Eq. (6.24)) as:

$$\theta^{(t+1)} = \theta^{(t)} + \rho(X^T y - X^T X \theta^{(t)}) \tag{6.40}$$

Given that the θ coordinates and the ψ coordinates are related via $\theta = A\psi + \theta^*$, we obtain:

$$A\psi^{(t+1)} = A\psi^{(t)} - \rho(X^T X A\psi^{(t)}), \tag{6.41}$$

where we have used the fact that θ^* solves the normal equations. Premultiplying both sides of this equation by A^T (recalling that A is orthogonal), we obtain:

$$\begin{aligned} \psi^{(t+1)} &= \psi^{(t)} - \rho(A^T X^T X A\psi^{(t)}) &(6.42)\\ &= \psi^{(t)} - \rho\Lambda\psi^{(t)}. &(6.43) \end{aligned}$$

This equation represents a decoupled set of equations in the components of the ψ vector:

$$\psi^{i(t+1)} = (1 - \rho\lambda_i)\psi^{i(t)}, \tag{6.44}$$

which converges if $(1 - \rho\lambda_i)$ is less than one in absolute value. That is, we require:

$$\|1 - \rho\lambda_i\| < 1, \tag{6.45}$$

which is equivalent to:

$$0 < \rho < 2/\lambda_i. \tag{6.46}$$

Given that this must be true for all λ_i we have recovered the same condition for convergence as obtained in the previous section (Eq. (6.33)).

In the decoupled coordinate system, we see that convergence condition amounts to the condition that if the algorithm hops from one side of an axis of the ellipsoid to the other, it must end up no further away from the axis than when it started. The axis associated with the maximum eigenvalue puts the strongest constraint on the step size.

6.4.3 LMS and stochastic approximation

It is beyond the scope of the book to provide a detailed consideration of the sense in which the LMS algorithm (and related "on-line" algorithms) converges to a solution, and we will content ourselves with providing pointers to the literature on stochastic approximation where such issues are addressed.[5] To get some sense of the issues involved, however, note that the path taken by the LMS algorithm in the parameter space depends on the particular way in which the training set is ordered. There are many kinds of ordering that may arise practice; typical examples include: (1) the algorithm passes through the training set in a fixed order, (2) varying orderings are used for each pass through the training set, and (3) data points are selected randomly with replacement from the training set. Moreover, (4) in other cases there is no "training set"; rather, the data points arrive as a potentially infinite stream. Another set of issues arises when one considers the meaning of "convergence." If the step size ρ remains fixed then the algorithm "converges" only in a stochastic sense, and there are several kinds of stochastic convergence that one can consider. It is also possible to consider variants of LMS in which the step size decreases to zero; under certain conditions (certain rates of decrease of the step size) the algorithm can be shown to converge to a point. As should be clear, a full analysis of LMS is a subtle business, and fairly sophisticated mathematical tools are required to do justice to the problem.

[5] See the section on "Historical remarks and bibliography" at the end of the chapter.

6.5 Weighted least squares

In later chapters we will need to solve a generalization of least squares, in which each data point is accompanied by a "weight" w_n. Intuitively, large weights correspond to data points that are "important," and small weights correspond to data points that are "unimportant." Let us set up this *weighted least squares* problem and display the corresponding normal equations.

Consider a set of weights w_n for each $n = 1, \ldots, N$. Let us incorporate these weights into the cost function as follows:

$$J(\theta) = \frac{1}{2} \sum_{n=1}^{N} w_n (y_n - \theta^T x_n)^2, \tag{6.47}$$

We can write this cost function in matrix form by defining a diagonal matrix $W \triangleq \mathrm{diag}(w_1, w_2, \ldots, w_N)$ and writing:

$$J(\theta) = \frac{1}{2}(y - X\theta)^T W (y - X\theta), \tag{6.48}$$

where we see that the weight matrix W can be viewed as defining a new metric with which to measure errors.

To obtain a solution θ^*, we take the gradient of Eq. (6.48):

$$\nabla_\theta J = -X^T W y - X^T W X \theta. \tag{6.49}$$

and set to zero:

$$X^T W X \theta^* = X^T W y, \tag{6.50}$$

These equations are the normal equations for weighted least squares.

6.6 Probabilistic interpretation

Thus far we have avoided making any probabilistic interpretation of the linear model and the least squares cost function. Let us now return to the statistical framework of linear regression in Chapter 5 and endow the terms in the linear model with probability distributions.

In Chapter 5 we augmented the linearity assumption with the assumption that the errors ϵ_n are Gaussian random variables having zero mean and variance σ^2. This assumption implies that the conditional probability of y_n given x_n is Gaussian with mean $\theta^T x_n$:

$$p(y_n | x_n, \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(y_n - \theta^T x_n)^2 \right\}. \tag{6.51}$$

We assumed moreover that the y_n are independent and identically distributed, conditional on x_n. Thus the joint conditional distribution of the data y is obtained by taking the product of the individual conditional probabilities:

$$p(y | x, \theta) = \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \theta^T x_n)^2 \right\}. \tag{6.52}$$

Taking the logarithm and dropping the terms that do not depend on the parameter θ, we obtain the following expression for the log likelihood:

$$l(\theta; x, y) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \theta^T x_n)^2. \tag{6.53}$$

This log likelihood is equivalent to the least-squares cost function $J(\theta)$ in Eq. (6.10). In particular, maximizing the log likelihood with respect to θ is equivalent to minimizing the least-squares cost function.

What we have shown is that the assumptions of a Gaussian distribution and IID sampling imply—within a maximum likelihood framework—the minimization of the least-squares cost function. Moreover, the normal equations characterize the maximum likelihood solution to the linear regression problem.

We can view this result as providing support for the likelihood-based approach to parameter estimation. In particular, in imposing probabilistic assumptions on the linear model so as to obtain a likelihood function, we have imposed neither more nor less constraint on the problem than is required to obtain a well-posed deterministic problem in the constraint satisfaction formulation. In particular, in the latter formulation, we need to decide how to measure the magnitudes of the errors and how to combine these magnitudes. These decisions have correspondences in the probabilistic formulation, in particular the Gaussian assumption effectively determines the metric by which we measure the errors, and the IID assumption determines the way in which the errors are combined. Both formulations are useful. In particular the constraint satisfaction perspective has helped us to understand that the linear, IID, and Gaussian assumptions comprise a natural family, essentially reflecting a Euclidean geometry. The probabilistic perspective provides additional insight; in particular, a Gaussian distribution for the errors can be justified via the central limit theorem if it is the case that the error terms ϵ_n are decomposable into sums of many small random terms.

It is also worth noting that in the frequentist approach to estimation we are not restricted to likelihood-based methods. In particular, we can view the least-squares cost function as providing an "estimator" that can be evaluated with the usual frequentist criteria. That is, we can define the least-squares estimator of a parameter as a value that minimizes the least-squares cost function, whether or not the underlying probability model involves a Gaussian assumption. If the underlying model is Gaussian then the least-squares approach and maximum likelihood coincide, but in general they can be viewed as competitors. The fact that least-squares estimates involve the solutions of systems of linear equations is a computational argument in their favor.

Although the geometric perspective provides significant insight in the case of the linear model, the probabilistic, likelihood-based perspective becomes increasingly powerful when we consider various generalizations of the linear model. For example, discrete variables are naturally handled by likelihood-based methods, as are hybrid models that involve combinations of discrete and continuous variables. Moreover, latent variables allow us to build more complex error models and Markov chains allow us to move beyond the IID assumption. Likelihood-based methods will be our focus throughout the remainder of the book.

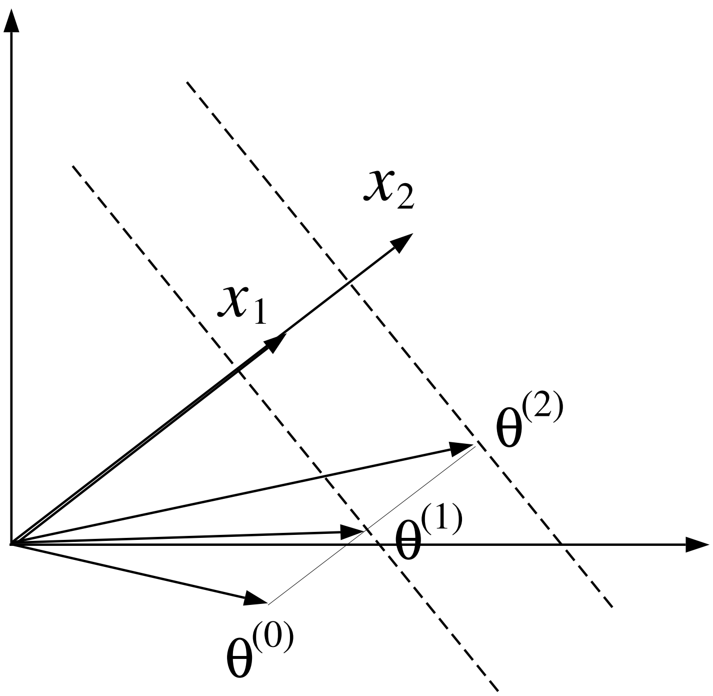

Figure 6.6: The geometry associated with the LMS algorithm in the case of two redundant input vectors x_1 and x_2. The dashed lines represent lines of solutions corresponding to each of the input vectors. There is a line of least-squares solutions that lies halfway between these two lines. The component of the initial parameter vector $\theta^{(0)}$ that is orthogonal to these lines does not vanish as the algorithm iterates.

6.7 Ridge regression

6.8 Sequential Bayesian methods

6.9 Historical remarks and bibliography