9 Performance Optimization

Objectives

We initiated our discussion of performance optimization in Chapter 8. There we introduced the Taylor series expansion as a tool for analyzing the performance surface, and then used it to determine conditions that must be satisfied by optimum points. In this chapter we will again use the Taylor series expansion, in this case to develop algorithms to locate the optimum points. We will discuss three different categories of optimization algorithm: steepest descent, Newton's method and conjugate gradient. In Chapters 10–14 we will apply all of these algorithms to the training of neural networks.

Theory and Examples

In the previous chapter we began our investigation of performance surfaces. Now we are in a position to develop algorithms to search the parameter space and locate minimum points of the surface (find the optimum weights and biases for a given neural network).

It is interesting to note that most of the algorithms presented in this chapter were developed hundreds of years ago. The basic principles of optimization were discovered during the 17th century, by such scientists and mathematicians as Kepler, Fermat, Newton and Leibniz. From 1950 on, these principles were rediscovered to be implemented on "high speed" (in comparison to the pen and paper available to Newton) digital computers. The success of these efforts stimulated significant research on new algorithms, and the field of optimization theory became recognized as a major branch of mathematics. Now neural network researchers have access to a vast storehouse of optimization theory and practice that can be applied to the training of neural networks.

The objective of this chapter, then, is to develop algorithms to optimize a performance index $F(\mathbf{x})$. For our purposes the word "optimize" will mean to find the value of $\mathbf{x}$ that minimizes $F(\mathbf{x})$. All of the optimization algorithms we will discuss are iterative. We begin from some initial guess, $\mathbf{x}_0$, and then update our guess in stages according to an equation of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k , \tag{9.1}$$

or

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k , \tag{9.2}$$

where the vector $\mathbf{p}_k$ represents a search direction, and the positive scalar α_k is the learning rate, which determines the length of the step.

The algorithms we will discuss in this chapter are distinguished by the choice of the search direction, $\mathbf{p}_k$. We will discuss three different possibilities. There are also a variety of ways to select the learning rate, α_k, and we will discuss several of these.

Steepest Descent

When we update our guess of the optimum (minimum) point using Eq. (9.1), we would like to have the function decrease at each iteration. In other words,

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k) . \tag{9.3}$$

How can we choose a direction, $\mathbf{p}_k$, so that for sufficiently small learning rate, α_k, we will move "downhill" in this way? Consider the first-order Taylor series expansion (see Eq. (8.9)) of $F(\mathbf{x})$ about the old guess $\mathbf{x}_k$:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k , \tag{9.4}$$

where $\mathbf{g}_k$ is the gradient evaluated at the old guess $\mathbf{x}_k$:

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_k} . \tag{9.5}$$

For $F(\mathbf{x}_{k+1})$ to be less than $F(\mathbf{x}_k)$, the second term on the right-hand side of Eq. (9.4) must be negative:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0 . \tag{9.6}$$

We will select an α_k that is small, but greater than zero. This implies:

$$\mathbf{g}_k^T \mathbf{p}_k < 0 . \tag{9.7}$$

Descent Direction

Any vector $\mathbf{p}_k$ that satisfies this equation is called a *descent direction*. The function must go down if we take a small enough step in this direction. This brings up another question. What is the direction of steepest descent? (In what direction will the function decrease most rapidly?) This will occur when

$$\mathbf{g}_k^T \mathbf{p}_k \tag{9.8}$$

is most negative. (We assume that the length of $\mathbf{p}_k$ does not change, only the direction.) This is an inner product between the gradient and the direction vector. It will be most negative when the direction vector is the negative of the gradient. (Review our discussion of directional derivatives on page 8-6.) Therefore a vector that points in the steepest descent direction is

$$\mathbf{p}_k = -\mathbf{g}_k . \tag{9.9}$$

Steepest Descent

Using this in the iteration of Eq. (9.1) produces the method of *steepest descent*:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k . \tag{9.10}$$

Learning Rate

For steepest descent there are two general methods for determining the *learning rate*, α_k. One approach is to minimize the performance index $F(\mathbf{x})$ with respect to α_k at each iteration. In this case we are minimizing along the line

$$\mathbf{x}_k - \alpha_k \mathbf{g}_k . \tag{9.11}$$

The other method for selecting α_k is to use a fixed value (e.g., $\alpha_k = 0.02$), or to use variable, but predetermined, values (e.g., $\alpha_k = 1/k$). We will discuss the choice of α_k in more detail in the following examples.

Let's apply the steepest descent algorithm to the following function,

$$F(\mathbf{x}) = x_1^2 + 25x_2^2, \tag{9.12}$$

starting from the initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}. \tag{9.13}$$

The first step is to find the gradient:

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix}. \tag{9.14}$$

If we evaluate the gradient at the initial guess we find

$$\mathbf{g}_0 = \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 1 \\ 25 \end{bmatrix}. \tag{9.15}$$

Assume that we use a fixed learning rate of $\alpha = 0.01$. The first iteration of the steepest descent algorithm would be

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.01 \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix}. \tag{9.16}$$

The second iteration of steepest descent produces

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix} - 0.01 \begin{bmatrix} 0.98 \\ 12.5 \end{bmatrix} = \begin{bmatrix} 0.4802 \\ 0.125 \end{bmatrix}. \tag{9.17}$$

If we continue the iterations we obtain the trajectory illustrated in Figure 9.1.

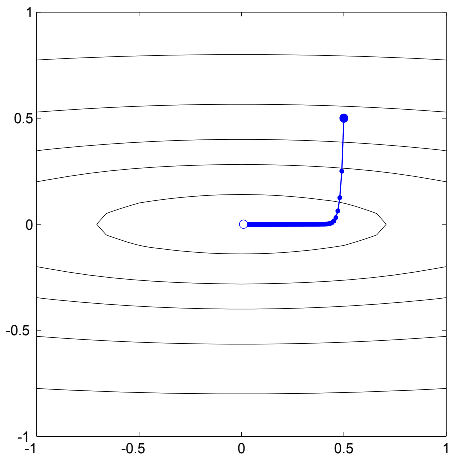

Figure 9.1 Trajectory for Steepest Descent with $\alpha = 0.01$

Note that the steepest descent trajectory, for small learning rate, follows a path that is always orthogonal to the contour lines. This is because the gradient is orthogonal to the contour lines. (See the discussion on page 8-6.)

How would a change in the learning rate change the performance of the algorithm? If we increase the learning rate to $\alpha = 0.035$, we obtain the trajectory illustrated in Figure 9.2. Note that the trajectory now oscillates. If we make the learning rate too large the algorithm will become unstable; the oscillations will increase instead of decaying.

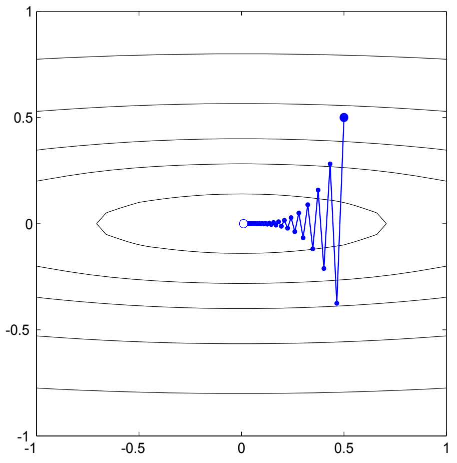

Figure 9.2 Trajectory for Steepest Descent with $\alpha = 0.035$

We would like to make the learning rate large, since then we will be taking large steps and would expect to converge faster. However, as we can see from this example, if we make the learning rate too large the algorithm will become unstable. Is there some way to predict the maximum allowable learning rate? This is not possible for arbitrary functions, but for quadratic functions we can set an upper limit.

Stable Learning Rates

Suppose that the performance index is a quadratic function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c \,. \tag{9.18}$$

From Eq. (8.38) the gradient of the quadratic function is

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} \,. \tag{9.19}$$

If we now insert this expression into our expression for the steepest descent algorithm (assuming a constant learning rate), we obtain

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{g}_k = \mathbf{x}_k - \alpha(\mathbf{A}\mathbf{x}_k + \mathbf{d}) \tag{9.20}$$

or

$$\mathbf{x}_{k+1} = [\mathbf{I} - \alpha\mathbf{A}]\mathbf{x}_k - \alpha\mathbf{d} \,. \tag{9.21}$$

This is a linear dynamic system, which will be stable if the eigenvalues of the matrix $[\mathbf{I} - \alpha\mathbf{A}]$ are less than one in magnitude (see [Brog91]). We can express the eigenvalues of this matrix in terms of the eigenvalues of the Hessian matrix $\mathbf{A}$. Let $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ and $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$ be the eigenvalues and eigenvectors of the Hessian matrix. Then

$$[\mathbf{I} - \alpha\mathbf{A}]\mathbf{z}_i = \mathbf{z}_i - \alpha\mathbf{A}\mathbf{z}_i = \mathbf{z}_i - \alpha\lambda_i\mathbf{z}_i = (1 - \alpha\lambda_i)\mathbf{z}_i \,. \tag{9.22}$$

Therefore the eigenvectors of $[\mathbf{I} - \alpha\mathbf{A}]$ are the same as the eigenvectors of $\mathbf{A}$, and the eigenvalues of $[\mathbf{I} - \alpha\mathbf{A}]$ are $(1 - \alpha\lambda_i)$. Our condition for the stability of the steepest descent algorithm is then

$$|(1 - \alpha\lambda_i)| < 1 \,. \tag{9.23}$$

If we assume that the quadratic function has a strong minimum point, then its eigenvalues must be positive numbers. Eq. (9.23) then reduces to

$$\alpha < \frac{2}{\lambda_i} \,. \tag{9.24}$$

Since this must be true for all the eigenvalues of the Hessian matrix we have

$$\alpha < \frac{2}{\lambda_{max}} \,. \tag{9.25}$$

The maximum stable learning rate is inversely proportional to the maximum curvature of the quadratic function. The curvature tells us how fast the gradient is changing. If the gradient is changing too fast we may jump past the minimum point so far that the gradient at the new location will be larger in magnitude (but opposite direction) than the gradient at the old location. This will cause the steps to increase in size at each iteration.

Let's apply this result to our previous example. The Hessian matrix for that quadratic function is

$$\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}. \tag{9.26}$$

The eigenvalues and eigenvectors of $\mathbf{A}$ are

$$\left\{ (\lambda_1 = 2), \left(\mathbf{z}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \right\}, \left\{ (\lambda_2 = 50), \left(\mathbf{z}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) \right\}. \tag{9.27}$$

Therefore the maximum allowable learning rate is

$$\alpha < \frac{2}{\lambda_{max}} = \frac{2}{50} = 0.04 . \tag{9.28}$$

This result is illustrated experimentally in Figure 9.3, which shows the steepest descent trajectories when the learning rate is just below ($\alpha = 0.039$) and just above ($\alpha = 0.041$), the maximum stable value.

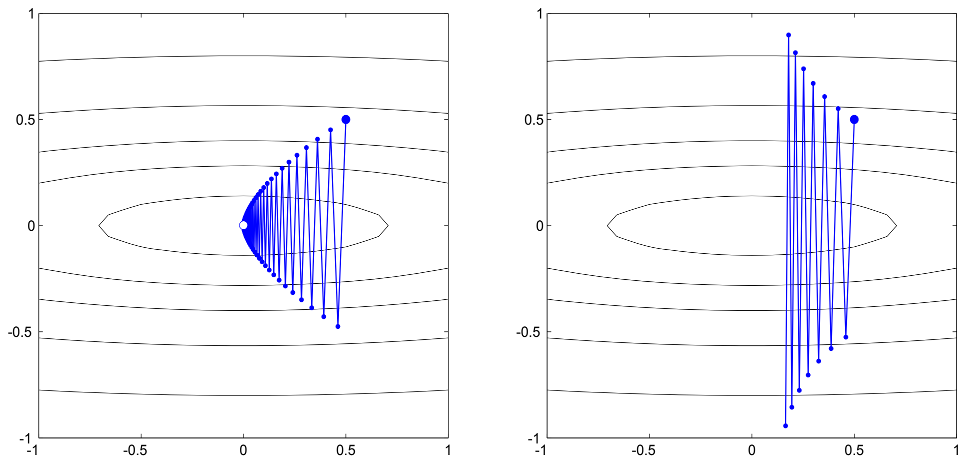

Figure 9.3 Trajectories for $\alpha = 0.039$ (left) and $\alpha = 0.041$ (right).

This example has illustrated several points. The learning rate is limited by the largest eigenvalue (second derivative) of the Hessian matrix. The algorithm tends to converge most quickly in the direction of the eigenvector corresponding to this largest eigenvalue, and we don't want to overshoot the minimum point by too far in that direction. (Note that in our examples the initial step is almost parallel to the x_2 axis, which is $\mathbf{z}_2$.) However, the algorithm will tend to converge most slowly in the direction of the eigenvector that corresponds to the smallest eigenvalue ($\mathbf{z}_1$ for our example). In the end it is the smallest eigenvalue, in combination with the learning rate, that determines how quickly the algorithm will converge. When there is a great difference in magnitude between the largest and smallest eigenvalues, the steepest descent algorithm will converge slowly.

To experiment with steepest descent on this quadratic function, use the Neural Network Design Demonstration Steepest Descent for a Quadratic (`nnd9sdq`).

Minimizing Along a Line

Another approach for selecting the learning rate is to minimize the performance index with respect to α_k at each iteration. In other words, choose α_k to minimize

$$F(\mathbf{x}_k + \alpha_k \mathbf{p}_k) . \tag{9.29}$$

To do this for arbitrary functions requires a line search, which we will discuss in Chapter 12. For quadratic functions it is possible to perform the linear minimization analytically. The derivative of Eq. (9.29) with respect to α_k, for quadratic $F(\mathbf{x})$, can be shown to be

$$\frac{d}{d\alpha_k} F(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \nabla F(\mathbf{x})^T \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k . \tag{9.30}$$

If we set this derivative equal to zero and solve for α_k, we obtain

$$\alpha_k = -\frac{\nabla F(\mathbf{x})^T \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k} , \tag{9.31}$$

where $\mathbf{A}_k$ is the Hessian matrix evaluated at the old guess $\mathbf{x}_k$:

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} . \tag{9.32}$$

(For quadratic functions the Hessian matrix is not a function of k.)

Let's apply steepest descent with line minimization to the following quadratic function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{x} , \tag{9.33}$$

starting from the initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 0.8 \\ -0.25 \end{bmatrix}. \tag{9.34}$$

The gradient of this function is

$$\nabla F(\mathbf{x}) = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{bmatrix}. \tag{9.35}$$

The search direction for steepest descent is the negative of the gradient. For the first iteration this will be

$$\mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}. \tag{9.36}$$

From Eq. (9.31), the learning rate for the first iteration will be

$$\alpha_0 = -\frac{\begin{bmatrix} 1.35 & 0.3 \end{bmatrix}\begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}}{\begin{bmatrix} -1.35 & -0.3 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}} = 0.413\,. \tag{9.37}$$

The first step of steepest descent will then produce

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0\mathbf{g}_0 = \begin{bmatrix} 0.8 \\ -0.25 \end{bmatrix} - 0.413\begin{bmatrix} 1.35 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.24 \\ -0.37 \end{bmatrix}. \tag{9.38}$$

The first five iterations of the algorithm are illustrated in Figure 9.4.

Note that the successive steps of the algorithm are orthogonal. Why does this happen? First, when we minimize along a line we will always stop at a point that is tangent to a contour line. Then, since the gradient is orthogonal to the contour line, the next step, which is along the negative of the gradient, will be orthogonal to the previous step.

We can show this analytically by using the chain rule on Eq. (9.30):

$$\frac{d}{d\alpha_k}F(\mathbf{x}_k + \alpha_k\mathbf{p}_k) = \frac{d}{d\alpha_k}F(\mathbf{x}_{k+1}) = \nabla F(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_{k+1}}\frac{d}{d\alpha_k}[\mathbf{x}_k + \alpha_k\mathbf{p}_k]$$

$$= \nabla F(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}_{k+1}}\mathbf{p}_k = \mathbf{g}_{k+1}^T\mathbf{p}_k\,. \tag{9.39}$$

Therefore at the minimum point, where this derivative is zero, the gradient is orthogonal to the previous search direction. Since the next search direction is the negative of this gradient, the consecutive search directions must be orthogonal. (Note that this result implies that when minimizing in any direction, the gradient at the minimum point will be orthogonal to the search direction, even if we are not using steepest descent. We will use this result in our discussion of conjugate directions.)

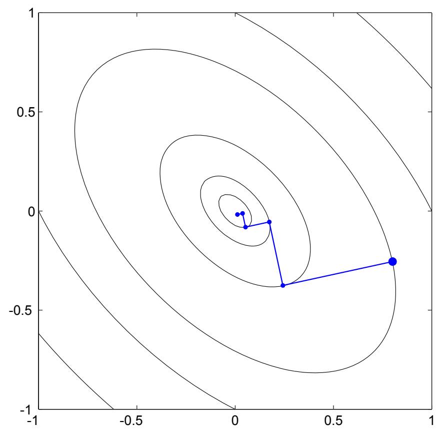

Figure 9.4 Steepest Descent with Minimization Along a Line

To experiment with steepest descent with minimization along a line, use the Neural Network Design Demonstration Method Comparison (`nnd9mc`).

Later in this chapter we will find that we can improve performance if we adjust the search directions, so that instead of being orthogonal they are *conjugate*. (We will define this term later.) If conjugate directions are used the function can be exactly minimized in at most n steps, where n is the dimension of $\mathbf{x}$. (There are certain types of quadratic functions that are minimized in one step by the steepest descent algorithm. Can you think of such a function? How is its Hessian matrix characterized?)

Newton's Method

The derivation of the steepest descent algorithm was based on the first-order Taylor series expansion (Eq. (9.4)). Newton's method is based on the second-order Taylor series:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T \mathbf{A}_k \Delta\mathbf{x}_k . \qquad (9.40)$$

The principle behind Newton's method is to locate the stationary point of this quadratic approximation to $F(\mathbf{x})$. If we use Eq. (8.38) to take the gradient of this quadratic function with respect to $\Delta\mathbf{x}_k$ and set it equal to zero, we find

$$\mathbf{g}_k + \mathbf{A}_k \Delta\mathbf{x}_k = \mathbf{0} . \tag{9.41}$$

Solving for $\Delta\mathbf{x}_k$ produces

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1}\mathbf{g}_k . \tag{9.42}$$

Newton's Method

Newton's method is then defined:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k . \tag{9.43}$$

To illustrate the operation of Newton's method, let's apply it to our previous example function of Eq. (9.12):

$$F(\mathbf{x}) = x_1^2 + 25x_2^2 . \tag{9.44}$$

The gradient and Hessian matrices are

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix} , \; \nabla^2 F(\mathbf{x}) = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix} . \tag{9.45}$$

If we start from the same initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} , \tag{9.46}$$

the first step of Newton's method would be

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} . \tag{9.47}$$

This method will always find the minimum of a quadratic function in one step. This is because Newton's method is designed to approximate a function as quadratic and then locate the stationary point of the quadratic approximation. If the original function is quadratic (with a strong minimum) it will be minimized in one step. The trajectory of Newton's method for this problem is given in Figure 9.5.

If the function $F(\mathbf{x})$ is not quadratic, then Newton's method will not generally converge in one step. In fact, we cannot be sure that it will converge at all, since this will depend on the function and the initial guess.

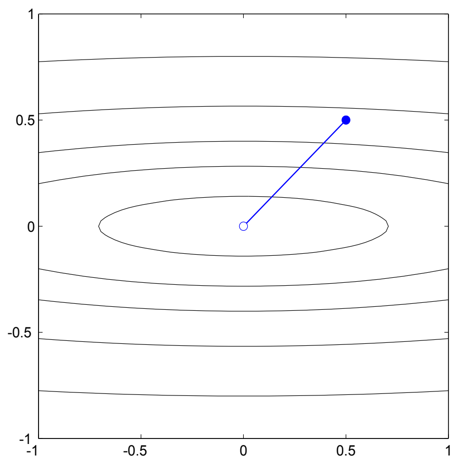

Figure 9.5 Trajectory for Newton's Method

Recall the function given by Eq. (8.18):

$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3\,. \tag{9.48}$$

We know from Chapter 8 (see Problem P8.5) that this function has three stationary points:

$$\mathbf{x}^1 = \begin{bmatrix} -0.41878 \\ 0.41878 \end{bmatrix},\ \mathbf{x}^2 = \begin{bmatrix} -0.134797 \\ 0.134797 \end{bmatrix},\ \mathbf{x}^3 = \begin{bmatrix} 0.55358 \\ -0.55358 \end{bmatrix}. \tag{9.49}$$

The first point is a strong local minimum, the second point is a saddle point, and the third point is a strong global minimum.

If we apply Newton's method to this problem, starting from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1.5 & 0 \end{bmatrix}^T$, our first iteration will be as shown in Figure 9.6. The graph on the left-hand side of the figure is a contour plot of the original function. On the right we see the quadratic approximation to the function at the initial guess.

The function is not minimized in one step, which is not surprising since the function is not quadratic. However, we do take a step toward the global minimum, and if we continue for two more iterations the algorithm will converge to within 0.01 of the global minimum. Newton's method converges quickly in many applications because analytic functions can be accurately approximated by quadratic functions in a small neighborhood of a strong minimum. So as we move closer to the minimum point, Newton's method will more accurately predict its location. In this case we can see that the contour plot of the quadratic approximation is similar to the contour plot of the original function near the initial guess.

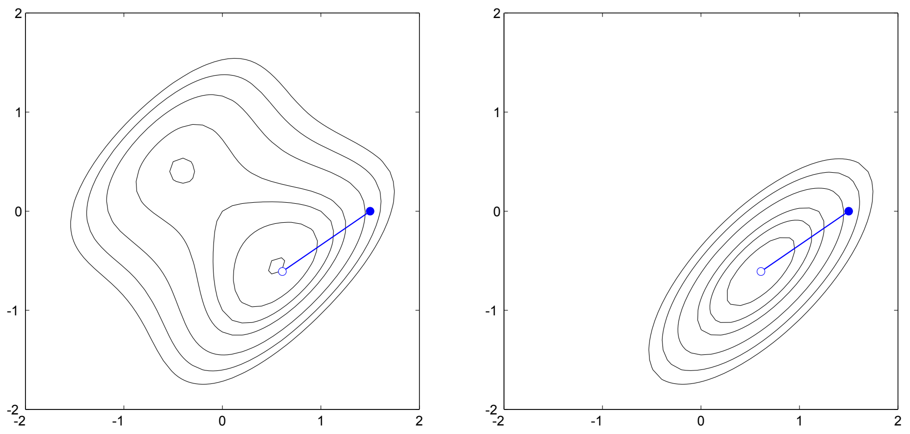

Figure 9.6 One Iteration of Newton's Method from $\mathbf{x}_0 = \begin{bmatrix}1.5 & 0\end{bmatrix}^T$

In Figure 9.7 we see one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix}-1.5 & 0\end{bmatrix}^T$. In this case we are converging to the local minimum. Clearly Newton's method cannot distinguish between a local minimum and a global minimum, since it approximates the function as a quadratic, and the quadratic function can have only one minimum. Newton's method, like steepest descent, relies on the local features of the surface (the first and second derivatives). It cannot know the global character of the function.

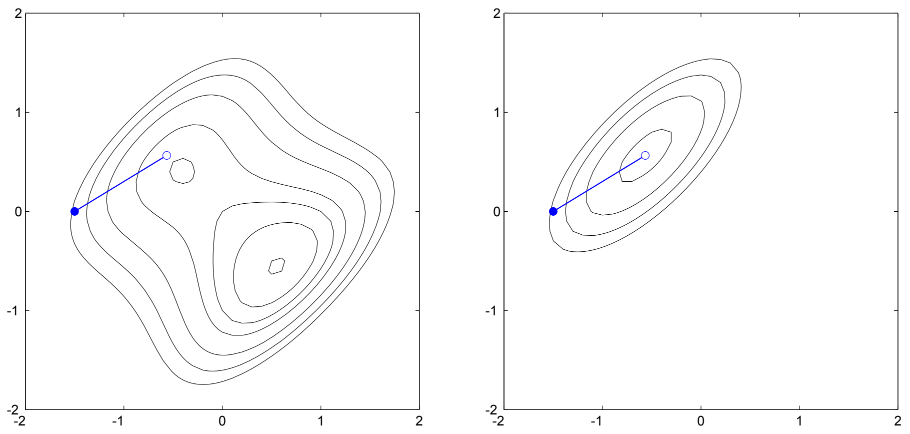

Figure 9.7 One Iteration of Newton's Method from $\mathbf{x}_0 = \begin{bmatrix}-1.5 & 0\end{bmatrix}^T$

In Figure 9.8 we see one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix}0.75 & 0.75\end{bmatrix}^T$. Now we are converging toward the saddle point of the function. Note that Newton's method locates the stationary point of the quadratic approximation to the function at the current guess. It does not distinguish between minima, maxima and saddle points. For this problem the quadratic approximation has a saddle point (indefinite Hessian matrix), which is near the saddle point of the original function. If we continue the iterations, the algorithm does converge to the saddle point of $F(\mathbf{x})$.

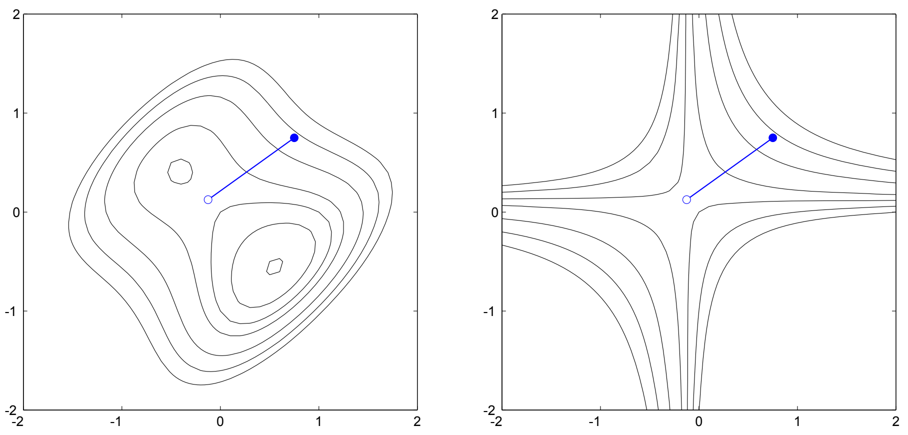

Figure 9.8 One Iteration of Newton's Method from $\mathbf{x}_0 = \begin{bmatrix} 0.75 & 0.75 \end{bmatrix}^T$

In each of the cases we have looked at so far the stationary point of the quadratic approximation has been close to a corresponding stationary point of $F(\mathbf{x})$. This is not always the case. In fact, Newton's method can produce very unpredictable results.

In Figure 9.9 we see one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1.15 & 0.75 \end{bmatrix}^T$. In this case the quadratic approximation predicts a saddle point, however, the saddle point is located very close to the local minimum of $F(\mathbf{x})$. If we continue the iterations, the algorithm will converge to the local minimum. Notice that the initial guess was actually farther away from the local minimum than it was for the previous case, in which the algorithm converged to the saddle point.

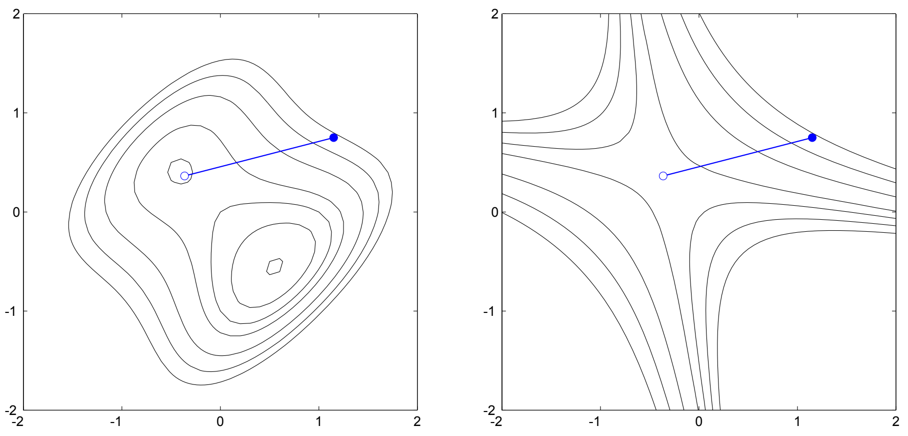

Figure 9.9 One Iteration of Newton's Method from $\mathbf{x}_0 = \begin{bmatrix} 1.15 & 0.75 \end{bmatrix}^T$

To experiment with Newton's method and steepest descent on this function, use the Neural Network Design Demonstrations Newton's Method (`nnd9nm`) *and* Steepest Descent (`nnd9sd`).

This is a good place to summarize some of the properties of Newton's method that we have observed.

While Newton's method usually produces faster convergence than steepest descent, the behavior of Newton's method can be quite complex. In addition to the problem of convergence to saddle points (which is very unlikely with steepest descent), it is possible for the algorithm to oscillate or diverge. Steepest descent is guaranteed to converge, if the learning rate is not too large or if we perform a linear minimization at each stage.

In Chapter 12 we will discuss a variation of Newton's method that is well suited to neural network training. It eliminates the divergence problem by using steepest descent steps whenever divergence begins to occur.

Another problem with Newton's method is that it requires the computation and storage of the Hessian matrix, as well as its inverse. If we compare steepest descent, Eq. (9.10), with Newton's method, Eq. (9.43), we see that their search directions will be identical when

$$\mathbf{A}_k = \mathbf{A}_k^{-1} = \mathbf{I} . \tag{9.50}$$

This observation has lead to a class of optimization algorithms know as quasi-Newton or one-step-secant methods. These methods replace $\mathbf{A}_k^{-1}$ with a positive definite matrix, $\mathbf{H}_k$, which is updated at each iteration without matrix inversion. The algorithms are typically designed so that for quadratic functions $\mathbf{H}_k$ will converge to $\mathbf{A}^{-1}$. (The Hessian is constant for quadratic functions.) See [Gill81], [Scal85] or [Batt92] for a discussion of these methods.

Conjugate Gradient

Quadratic Termination

Newton's method has a property called *quadratic termination*, which means that it minimizes a quadratic function exactly in a finite number of iterations. Unfortunately, it requires calculation and storage of the second derivatives. When the number of parameters, n, is large, it may be impractical to compute all of the second derivatives. (Note that the gradient has n elements, while the Hessian has n^2 elements.) This is especially true with neural networks, where practical applications can require several hundred to many thousand weights. For these cases we would like to have methods that require only first derivatives but still have quadratic termination.

Recall the performance of the steepest descent algorithm, with linear searches at each iteration. The search directions at consecutive iterations were orthogonal (see Figure 9.4). For quadratic functions with elliptical contours this produces a zig-zag trajectory of short steps. Perhaps quadratic search directions are not the best choice. Is there a set of search directions that will guarantee quadratic termination? One possibility is conjugate directions.

Suppose that we wish to locate the minimum of the following quadratic function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c \, . \tag{9.51}$$

Conjugate

A set of vectors $\{\mathbf{p}_k\}$ is mutually *conjugate* with respect to a positive definite Hessian matrix $\mathbf{A}$ if and only if

$$\mathbf{p}_k^T\mathbf{A}\mathbf{p}_j = 0 \qquad k \neq j \cdot \tag{9.52}$$

As with orthogonal vectors, there are an infinite number of mutually conjugate sets of vectors that span a given n-dimensional space. One set of conjugate vectors consists of the eigenvectors of $\mathbf{A}$. Let $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ and $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n\}$ be the eigenvalues and eigenvectors of the Hessian matrix. To see that the eigenvectors are conjugate, replace $\mathbf{p}_k$ with $\mathbf{z}_k$ in Eq. (9.52):

$$\mathbf{z}_k^T\mathbf{A}\mathbf{z}_j = \lambda_j\mathbf{z}_k^T\mathbf{z}_j = 0 \qquad k \neq j \, , \tag{9.53}$$

where the last equality holds because the eigenvectors of a symmetric matrix are mutually orthogonal. Therefore the eigenvectors are both conjugate and orthogonal. (Can you find a quadratic function where all orthogonal vectors are also conjugate?)

It is not surprising that we can minimize a quadratic function exactly by searching along the eigenvectors of the Hessian matrix, since they form the principal axes of the function contours. (See the discussion on pages 8-13 through 8-19.) Unfortunately this is not of much practical help, since to find the eigenvectors we must first find the Hessian matrix. We want to find an algorithm that does not require the computation of second derivatives.

It can be shown (see [Scal85] or [Gill81]) that if we make a sequence of exact linear searches along any set of conjugate directions $\{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$, then the exact minimum of any quadratic function, with n parameters, will be reached in at most n searches. The question is “How can we construct these conjugate search directions?” First, we want to restate the conjugacy condition, which is given in Eq. (9.52), without use of the Hessian matrix. Recall that for quadratic functions

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} \, , \tag{9.54}$$

$$\nabla^2 F(\mathbf{x}) = \mathbf{A} \, . \tag{9.55}$$

By combining these equations we find that the change in the gradient at iteration $k+1$ is

$$\Delta \mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k = (\mathbf{A}\mathbf{x}_{k+1} + \mathbf{d}) - (\mathbf{A}\mathbf{x}_k + \mathbf{d}) = \mathbf{A}\Delta\mathbf{x}_k , \tag{9.56}$$

where, from Eq. (9.2), we have

$$\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k , \tag{9.57}$$

and α_k is chosen to minimize $F(\mathbf{x})$ in the direction $\mathbf{p}_k$.

We can now restate the conjugacy conditions (Eq. (9.52)):

$$\alpha_k \mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = \Delta\mathbf{x}_k^T \mathbf{A} \mathbf{p}_j = \Delta\mathbf{g}_k^T \mathbf{p}_j = 0 \qquad k \neq j \cdot \tag{9.58}$$

Note that we no longer need to know the Hessian matrix. We have restated the conjugacy conditions in terms of the changes in the gradient at successive iterations of the algorithm. The search directions will be conjugate if they are orthogonal to the changes in the gradient.

Note that the first search direction, $\mathbf{p}_0$, is arbitrary, and $\mathbf{p}_1$ can be any vector that is orthogonal to $\Delta\mathbf{g}_0$. Therefore there are an infinite number of sets of conjugate vectors. It is common to begin the search in the steepest descent direction:

$$\mathbf{p}_0 = -\mathbf{g}_0 . \tag{9.59}$$

Then, at each iteration we need to construct a vector $\mathbf{p}_k$ that is orthogonal to $\{\Delta\mathbf{g}_0, \Delta\mathbf{g}_1, \ldots, \Delta\mathbf{g}_{k-1}\}$. It is a procedure similar to Gram-Schmidt orthogonalization, which we discussed in Chapter 5. It can be simplified (see [Scal85]) to iterations of the form

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1} . \tag{9.60}$$

The scalars β_k can be chosen by several different methods, which produce equivalent results for quadratic functions. The most common choices (see [Scal85]) are

$$\beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta\mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} , \tag{9.61}$$

due to Hestenes and Stiefel,

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \tag{9.62}$$

due to Fletcher and Reeves, and

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \tag{9.63}$$

due to Polak and Ribiére.

Conjugate Gradient

To summarize our discussion, the *conjugate gradient* method consists of the following steps:

1. Select the first search direction to be the negative of the gradient, as in Eq. (9.59).
2. Take a step according to Eq. (9.57), selecting the learning rate α_k to minimize the function along the search direction. We will discuss general linear minimization techniques in Chapter 12. For quadratic functions we can use Eq. (9.31).
3. Select the next search direction according to Eq. (9.60), using Eq. (9.61), Eq. (9.62), or Eq. (9.63) to calculate β_k.
4. If the algorithm has not converged, return to step 2.

To illustrate the performance of the algorithm, recall the example we used to demonstrate steepest descent with linear minimization:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \mathbf{x}, \tag{9.64}$$

with initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 0.8 \\ -0.25 \end{bmatrix}. \tag{9.65}$$

The gradient of this function is

$$\nabla F(\mathbf{x}) = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + 2x_2 \end{bmatrix}. \tag{9.66}$$

As with steepest descent, the first search direction is the negative of the gradient:

$$\mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}. \tag{9.67}$$

From Eq. (9.31), the learning rate for the first iteration will be

$$\alpha_0 = -\frac{\begin{bmatrix} 1.35 & 0.3 \end{bmatrix}\begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}}{\begin{bmatrix} -1.35 & -0.3 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix}} = 0.413 . \tag{9.68}$$

The first step of conjugate gradient is therefore:

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{p}_0 = \begin{bmatrix} 0.8 \\ -0.25 \end{bmatrix} + 0.413 \begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix} = \begin{bmatrix} 0.24 \\ -0.37 \end{bmatrix}, \tag{9.69}$$

which is equivalent to the first step of steepest descent with minimization along a line.

Now we need to find the second search direction from Eq. (9.60). This requires the gradient at $\mathbf{x}_1$:

$$\mathbf{g}_1 = \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_1} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 0.24 \\ -0.37 \end{bmatrix} = \begin{bmatrix} 0.11 \\ -0.5 \end{bmatrix}. \tag{9.70}$$

We can now find β_1:

$$\beta_1 = \frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = \frac{\begin{bmatrix} 0.11 & -0.5 \end{bmatrix}\begin{bmatrix} 0.11 \\ -0.5 \end{bmatrix}}{\begin{bmatrix} 1.35 & 0.3 \end{bmatrix}\begin{bmatrix} 1.35 \\ 0.3 \end{bmatrix}} = \frac{0.2621}{1.9125} = 0.137 , \tag{9.71}$$

using the method of Fletcher and Reeves (Eq. (9.62)). The second search direction is then computed from Eq. (9.60):

$$\mathbf{p}_1 = -\mathbf{g}_1 + \beta_1 \mathbf{p}_0 = \begin{bmatrix} -0.11 \\ 0.5 \end{bmatrix} + 0.137 \begin{bmatrix} -1.35 \\ -0.3 \end{bmatrix} = \begin{bmatrix} -0.295 \\ 0.459 \end{bmatrix}. \tag{9.72}$$

From Eq. (9.31), the learning rate for the second iteration will be

$$\alpha_1 = -\frac{\begin{bmatrix} 0.11 & -0.5 \end{bmatrix}\begin{bmatrix} -0.295 \\ 0.459 \end{bmatrix}}{\begin{bmatrix} -0.295 & 0.459 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} -0.295 \\ 0.459 \end{bmatrix}} = \frac{0.262}{0.325} = 0.807 . \tag{9.73}$$

The second step of conjugate gradient is therefore

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} 0.24 \\ -0.37 \end{bmatrix} + 0.807 \begin{bmatrix} -0.295 \\ 0.459 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \qquad (9.74)$$

As predicted, the algorithm converges exactly to the minimum in two iterations (since this is a two-dimensional quadratic function), as illustrated in Figure 9.10. Compare this result with the steepest descent algorithm, as shown in Figure 9.4. The conjugate gradient algorithm adjusts the second search direction so that it will pass through the minimum of the function (center of the function contours), instead of using an orthogonal search direction, as in steepest descent.

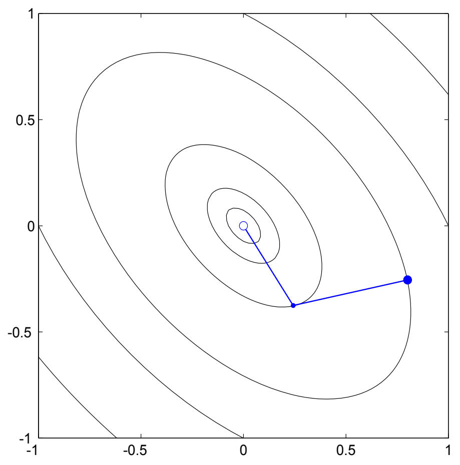

Figure 9.10 Conjugate Gradient Algorithm

We will return to the conjugate gradient algorithm in Chapter 12. In that chapter we will discuss how the algorithm should be adjusted for non-quadratic functions.

To experiment with the conjugate gradient algorithm and compare it with steepest descent, use the Neural Network Design Demonstration Method Comparison (`nnd9mc`).

Summary of Results

General Minimization Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

or

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$

Steepest Descent Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$$

$$\text{Where } \mathbf{g}_k \equiv \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_k}$$

Stable Learning Rate ($\alpha_k = \alpha$, constant)

$$\alpha < \frac{2}{\lambda_{max}}$$

$$\{\lambda_1, \lambda_2, \dots, \lambda_n\} \text{ Eigenvalues of Hessian matrix } \mathbf{A}$$

Learning Rate to Minimize Along the Line $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$$\alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k} \qquad \text{(For quadratic functions)}$$

After Minimizing Along the Line $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$

$$\mathbf{g}_{k+1}^T \mathbf{p}_k = 0$$

Newton's Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$$

$$\text{Where } \mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_k}$$

Conjugate Gradient Algorithm

$$\Delta \mathbf{x}_k = \alpha_k \mathbf{p}_k$$

Learning rate α_k is chosen to minimize along the line $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$.

$$\mathbf{p}_0 = -\mathbf{g}_0$$

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta \mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \text{ or } \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \text{ or } \beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Where $\mathbf{g}_k \equiv \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_k}$ and $\Delta \mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$.

Solved Problems

P9.1 We want to find the minimum of the following function:

$$F(\mathbf{x}) = 5x_1^2 - 6x_1x_2 + 5x_2^2 + 4x_1 + 4x_2 .$$

i. Sketch a contour plot of this function.

ii. Sketch the trajectory of the steepest descent algorithm on the contour plot of part (i) if the initial guess is $\mathbf{x}_0 = \begin{bmatrix} -1 & -2.5 \end{bmatrix}^T$. Assume a very small learning rate is used.

iii. What is the maximum stable learning rate?

i. To sketch the contour plot we first need to find the Hessian matrix. For quadratic functions we can do this by putting the function into the standard form (see Eq. (8.35)):

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c = \frac{1}{2}\mathbf{x}^T\begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}\mathbf{x} + \begin{bmatrix} 4 & 4 \end{bmatrix}\mathbf{x} .$$

From Eq. (8.39) the Hessian matrix is

$$\nabla^2 F(\mathbf{x}) = \mathbf{A} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} .$$

The eigenvalues and eigenvectors of this matrix are

$$\lambda_1 = 4 , \; \mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} , \; \lambda_2 = 16 , \; \mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix} .$$

From the discussion on quadratic functions in Chapter 8 (see page 8-15) we know that the function contours are elliptical. The maximum curvature of $F(\mathbf{x})$ is in the direction of $\mathbf{z}_2$, since λ_2 is larger than λ_1, and the minimum curvature is in the direction of $\mathbf{z}_1$ (the long axis of the ellipses).

Next we need to find the center of the contours (the stationary point). This occurs when the gradient is equal to zero. From Eq. (8.38) we find

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}\mathbf{x} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} .$$

Therefore

$$\mathbf{x}^* = -\begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

The contours will be elliptical, centered at $\mathbf{x}^*$, with long axis in the direction of $\mathbf{z}_1$. The contour plot is shown in Figure P9.1.

ii. We know that the gradient is always orthogonal to the contour line, therefore the steepest descent trajectory, if we take small enough steps, will follow a path that is orthogonal to each contour line it intersects. We can therefore trace the trajectory without performing any computations. The result is shown in Figure P9.1.

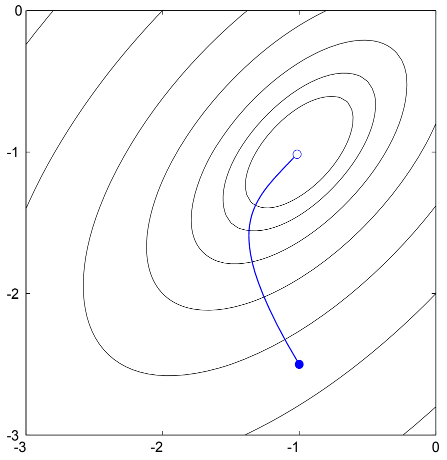

Figure P9.1 Contour Plot and Steep. Desc. Trajectory for Problem P9.1

iii. From Eq. (9.25) we know that the maximum stable learning rate for a quadratic function is determined by the maximum eigenvalue of the Hessian matrix:

$$\alpha < \frac{2}{\lambda_{max}}.$$

The maximum eigenvalue for this problem is $\lambda_2 = 16$, therefore for stability

$$\alpha < \frac{2}{16} = 0.125.$$

This result is verified experimentally in Figure P9.2, which shows the steepest descent trajectories when the learning rate is just below ($\alpha = 0.12$) and just above ($\alpha = 0.13$) the maximum stable value.

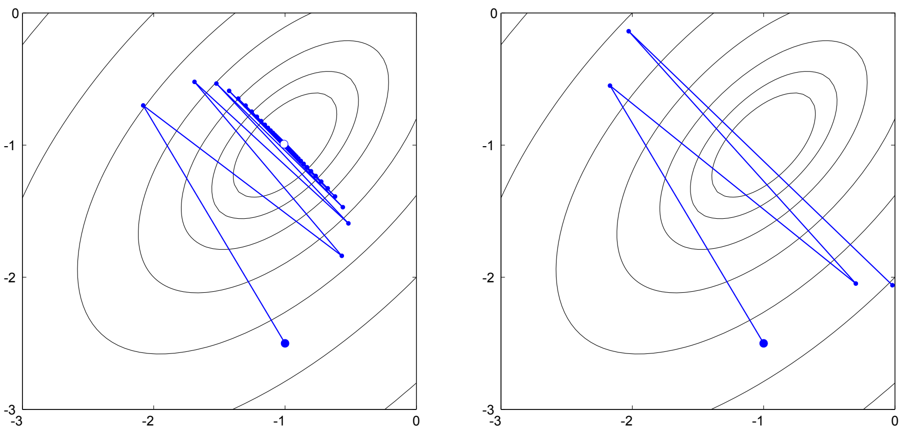

Figure P9.2 Trajectories for $\alpha = 0.12$ (left) and $\alpha = 0.13$ (right)

P9.2 Consider again the quadratic function of Problem P9.1. Take two steps of the steepest descent algorithm, minimizing along a line at each step. Use the following initial condition:

$$\mathbf{x}_0 = \begin{bmatrix} 0 & -2 \end{bmatrix}^T .$$

In Problem P9.1 we found the gradient of the function to be

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} .$$

If we evaluate this at $\mathbf{x}_0$, we find

$$\mathbf{g}_0 = \nabla F(\mathbf{x}_0) = \mathbf{A}\mathbf{x}_0 + \mathbf{d} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \begin{bmatrix} 0 \\ -2 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 16 \\ -16 \end{bmatrix} .$$

Therefore the first search direction is

$$\mathbf{p}_0 = -\mathbf{g}_0 = \begin{bmatrix} -16 \\ 16 \end{bmatrix} .$$

To minimize along a line, for a quadratic function, we can use Eq. (9.31):

$$\alpha_0 = -\frac{\mathbf{g}_0^T \mathbf{p}_0}{\mathbf{p}_0^T \mathbf{A} \mathbf{p}_0} = -\frac{\begin{bmatrix} 16 & -16 \end{bmatrix} \begin{bmatrix} -16 \\ 16 \end{bmatrix}}{\begin{bmatrix} -16 & 16 \end{bmatrix} \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \begin{bmatrix} -16 \\ 16 \end{bmatrix}} = -\frac{-512}{8192} = 0.0625 .$$

Therefore the first iteration of steepest descent will be

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{g}_0 = \begin{bmatrix} 0 \\ -2 \end{bmatrix} - 0.0625 \begin{bmatrix} 16 \\ -16 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}.$$

To begin the second iteration we need to find the gradient at $\mathbf{x}_1$:

$$\mathbf{g}_1 = \nabla F(\mathbf{x}_1) = \mathbf{A}\mathbf{x}_1 + \mathbf{d} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Therefore we have reached a stationary point; the algorithm has converged. From Problem P9.1 we know that $\mathbf{x}_1$ is indeed the minimum point of this quadratic function. The trajectory is shown in Figure P9.3.

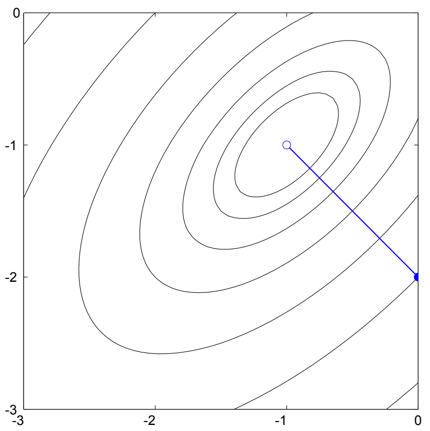

Figure P9.3 Steepest Descent with Linear Minimization for Problem P9.2

This is an unusual case, where the steepest descent algorithm located the minimum in one iteration. Notice that this occurred because the initial guess was located in the direction of one of the eigenvectors of the Hessian matrix, with respect to the minimum point. For those cases where every direction is an eigenvector, the steepest descent algorithm will always locate the minimum in one iteration. What would this imply about the eigenvalues of the Hessian matrix?

P9.3 Recall Problem P8.6, in which we derived a performance index for a linear neural network. The network, which is displayed again in Figure P9.4, was to be trained for the following input/output pairs:

$$\{(p_1 = 2), (t_1 = 0.5)\}, \{(p_2 = -1), (t_2 = 0)\}$$

The performance index for the network was defined to be

$$F(\mathbf{x}) = (t_1 - a_1(\mathbf{x}))^2 + (t_2 - a_2(\mathbf{x}))^2,$$

which was displayed in Figure P8.8.

i. Use the steepest descent algorithm to locate the optimal parameters for this network (recall that $\mathbf{x} = \begin{bmatrix} w & b \end{bmatrix}^T$), starting from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$. Use a learning rate of $\alpha = 0.05$.

ii. What is the maximum stable learning rate?

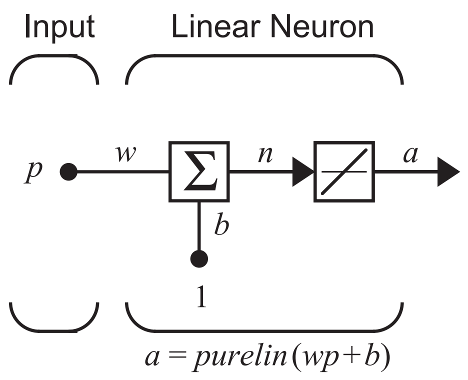

Figure P9.4 Linear Network for Problems P9.3 and P8.6

i. In Problem P8.6 we found that the performance index could be written in quadratic form:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c,$$

where

$$c = \mathbf{t}^T\mathbf{t} = \begin{bmatrix} 0.5 & 0 \end{bmatrix}\begin{bmatrix} 0.5 \\ 0 \end{bmatrix} = 0.25,$$

$$\mathbf{d} = -2\mathbf{G}^T\mathbf{t} = -2\begin{bmatrix} 2 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 0.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix},$$

$$\mathbf{A} = 2\mathbf{G}^T\mathbf{G} = \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix}.$$

The gradient at $\mathbf{x}_0$ is

$$\mathbf{g}_0 = \nabla F(\mathbf{x}_0) = \mathbf{A}\mathbf{x}_0 + \mathbf{d} = \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}.$$

The first iteration of steepest descent will be

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 0.05 \begin{bmatrix} 10 \\ 5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.75 \end{bmatrix}.$$

The second iteration will be

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.5 \\ 0.75 \end{bmatrix} - 0.05 \begin{bmatrix} 4.5 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.275 \\ 0.6 \end{bmatrix}.$$

The remaining iterations are displayed in Figure P9.5. The algorithm converges to the minimum point $\mathbf{x}^* = \begin{bmatrix} 0.167 & 0.167 \end{bmatrix}^T$. Therefore the optimal value for both the weight and the bias of this network is 0.167.

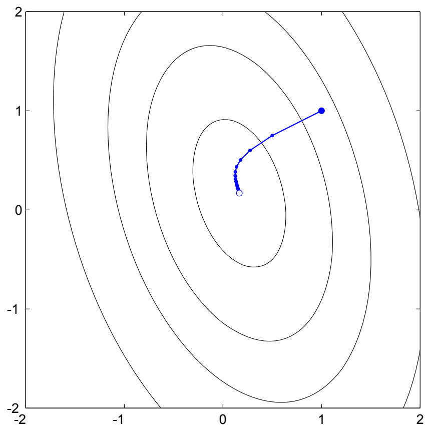

Figure P9.5 Steepest Descent Trajectory for Problem P9.3 with $\alpha = 0.05$

Note that in order to train this network we needed to know all of the input/output pairs. We then performed iterations of the steepest descent algorithm until convergence was achieved. In Chapter 10 we will introduce an adaptive algorithm, based on steepest descent, for training linear networks. With this adaptive algorithm the network parameters are updated after each input/output pair is presented. We will show how this allows the network to adapt to a changing environment.

ii. The maximum eigenvalue of the Hessian matrix for this problem is $\lambda_1 = 10.6$ (see Problem P8.6), therefore for stability

$$\alpha < \frac{2}{10.6} = 0.1887.$$

P9.4 Consider the function

$$F(\mathbf{x}) = e^{(x_1^2 - x_1 + 2x_2^2 + 4)}.$$

Take one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1 & -2 \end{bmatrix}^T$. How close is this result to the minimum point of $F(\mathbf{x})$? Explain.

The first step is to find the gradient and the Hessian matrix. The gradient is given by

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = e^{(x_1^2 - x_1 + 2x_2^2 + 4)} \begin{bmatrix} (2x_1 - 1) \\ (4x_2) \end{bmatrix},$$

and the Hessian matrix is given by

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) \end{bmatrix}$$

$$= e^{(x_1^2 - x_1 + 2x_2^2 + 4)} \begin{bmatrix} 4x_1^2 - 4x_1 + 3 & (2x_1 - 1)(4x_2) \\ (2x_1 - 1)(4x_2) & 16x_2^2 + 4 \end{bmatrix}$$

If we evaluate these at the initial guess we find

$$\mathbf{g}_0 = \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 0.163 \times 10^6 \\ -1.302 \times 10^6 \end{bmatrix},$$

and

$$\mathbf{A}_0 = \nabla^2 F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 0.049 \times 10^7 & -0.130 \times 10^7 \\ -0.130 \times 10^7 & 1.107 \times 10^7 \end{bmatrix}.$$

Therefore the first iteration of Newton's method, from Eq. (9.43), will be

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{A}_0^{-1}\mathbf{g}_0 = \begin{bmatrix} 1 \\ -2 \end{bmatrix} - \begin{bmatrix} 0.049\times10^7 & -0.130\times10^7 \\ -0.130\times10^7 & 1.107\times10^7 \end{bmatrix}^{-1} \begin{bmatrix} 0.163\times10^6 \\ -1.302\times10^6 \end{bmatrix} = \begin{bmatrix} 0.971 \\ -1.886 \end{bmatrix}$$

How close is this to the true minimum point of $F(\mathbf{x})$? First, note that the exponent of $F(\mathbf{x})$ is a quadratic function:

$$x_1^2 - x_1 + 2x_2^2 + 4 = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{d}^T\mathbf{x} + c = \frac{1}{2}\mathbf{x}^T\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}\mathbf{x} + \begin{bmatrix} -1 & 0 \end{bmatrix}\mathbf{x} + 4 .$$

The minimum point of $F(\mathbf{x})$ will be the same as the minimum point of the exponent, which is

$$\mathbf{x}^* = -\mathbf{A}^{-1}\mathbf{d} = -\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}^{-1}\begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} .$$

Therefore Newton's method has taken only a very small step toward the true minimum point. This is because $F(\mathbf{x})$ cannot be accurately approximated by a quadratic function in the neighborhood of $\mathbf{x}_0 = \begin{bmatrix} 1 & -2 \end{bmatrix}^T$.

For this problem Newton's method will converge to the true minimum point, but it will take many iterations. The trajectory for Newton's method is illustrated in Figure P9.6.

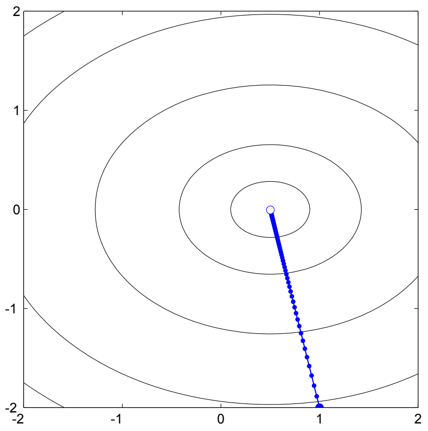

Figure P9.6 Newton's Method Trajectory for Problem P9.4

P9.5 Compare the performance of Newton's method and steepest descent on the following function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{x} .$$

Start from the initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} .$$

Recall that this function is an example of a stationary valley (see Eq. (8.59) and Figure 8.9). The gradient is

$$\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{d} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \mathbf{x}$$

and the Hessian matrix is

$$\nabla^2 F(\mathbf{x}) = \mathbf{A} = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} .$$

Newton's method is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k .$$

Note, however, that we cannot actually perform this algorithm, because the Hessian matrix is singular. We know from our discussion of this function in Chapter 8 that this function does not have a strong minimum, but it does have a weak minimum along the line $x_1 = x_2$.

What about steepest descent? If we start from the initial guess, with learning rate $\alpha = 0.1$, the first two iterations will be

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha\mathbf{g}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} ,$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha\mathbf{g}_1 = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} - 0.1 \begin{bmatrix} 0.8 \\ -0.8 \end{bmatrix} = \begin{bmatrix} 0.82 \\ 0.18 \end{bmatrix} .$$

The complete trajectory is shown in Figure P9.7. This is a case where the steepest descent algorithm performs better than Newton's method. Steepest descent converges to a minimum point (weak minimum), while Newton's method fails to converge. In Chapter 12 we will discuss a technique that combines steepest descent with Newton's method, to overcome the problem of singular (or almost singular) Hessian matrices.

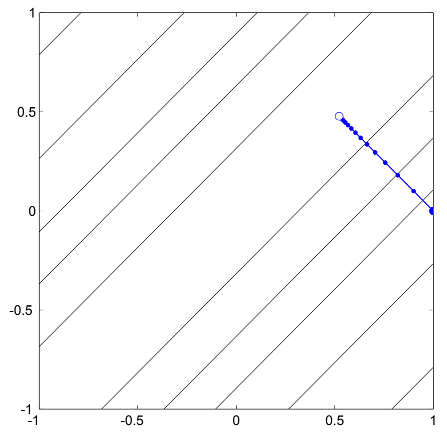

Figure P9.7 Steepest Descent Trajectory for Problem P9.5 with $\alpha = 0.1$

P9.6 Consider the following function:

$$F(\mathbf{x}) = x_1^3 + x_1x_2 - x_1^2x_2^2$$

i. Perform one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

ii. Find the second-order Taylor series expansion of $F(\mathbf{x})$ about $\mathbf{x}_0$. Is this quadratic function minimized at the point $\mathbf{x}_1$ found in part (i)? Explain.

i. The gradient of $F(\mathbf{x})$ is

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 3x_1^2 + x_2 - 2x_1x_2^2 \\ x_1 - 2x_1^2x_2 \end{bmatrix},$$

and the Hessian matrix is

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} 6x_1 - 2x_2^2 & 1 - 4x_1x_2 \\ 1 - 4x_1x_2 & -2x_1^2 \end{bmatrix}.$$

If we evaluate these at the initial guess we find

$$\mathbf{g}_0 = \nabla F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 2 \\ -1 \end{bmatrix},$$

$$\mathbf{A}_0 = \nabla^2 F(\mathbf{x})\big|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 4 & -3 \\ -3 & -2 \end{bmatrix}.$$

The first iteration of Newton's method is then

$$\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{A}_0^{-1}\mathbf{g}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 4 & -3 \\ -3 & -2 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.5882 \\ 1.1176 \end{bmatrix}.$$

ii. From Eq. (9.40), the second-order Taylor series expansion of $F(\mathbf{x})$ about $\mathbf{x}_0$ is

$$F(\mathbf{x}) = F(\mathbf{x}_0 + \Delta\mathbf{x}_0) \approx F(\mathbf{x}_0) + \mathbf{g}_0^T \Delta\mathbf{x}_0 + \frac{1}{2}\Delta\mathbf{x}_0^T \mathbf{A}_0 \Delta\mathbf{x}_0 .$$

If we substitute the values for $\mathbf{x}_0$, $\mathbf{g}_0$ and $\mathbf{A}_0$, we find

$$F(\mathbf{x}) \approx 1 + \begin{bmatrix} 2 & -1 \end{bmatrix} \left\{ \mathbf{x} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\} + \frac{1}{2} \left\{ \mathbf{x} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}^T \begin{bmatrix} 4 & -3 \\ -3 & -2 \end{bmatrix} \left\{ \mathbf{x} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

This can be reduced to

$$F(\mathbf{x}) \approx -2 + \begin{bmatrix} 1 & 4 \end{bmatrix} \mathbf{x} + \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 4 & -3 \\ -3 & -2 \end{bmatrix} \mathbf{x} .$$

This function has a stationary point at $\mathbf{x}_1$. The question is whether or not the stationary point is a strong minimum. This can be determined from the eigenvalues of the Hessian matrix. If both eigenvalues are positive, it is a strong minimum. If both eigenvalues are negative, it is a strong maximum. If the two eigenvalues have opposite signs, it is a saddle point. In this case the eigenvalues of $\mathbf{A}_0$ are

$$\lambda_1 = 5.24 \text{ and } \lambda_2 = -3.24 .$$

Therefore the quadratic approximation to $F(\mathbf{x})$ at $\mathbf{x}_0$ is not minimized at $\mathbf{x}_1$, since it is a saddle point. Figure P9.8 displays the contour plots of $F(\mathbf{x})$ and its quadratic approximation.

This sort of problem was also illustrated in Figure 9.8 and Figure 9.9. Newton's method does locate the stationary point of the quadratic approximation of the function at the current guess. It does not distinguish between minima, maxima and saddle points.

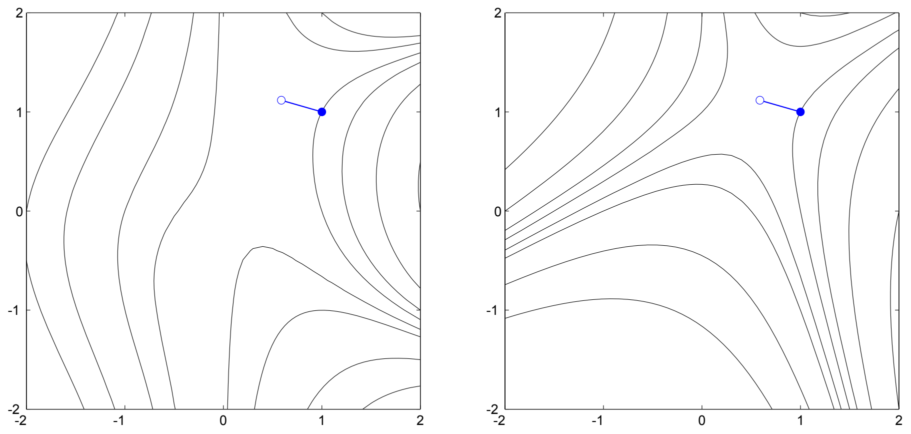

Figure P9.8 One Iteration of Newton's Method from $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$

P9.7 Repeat Problem P9.3 (i) using the conjugate gradient algorithm.

Recall that the function to be minimized was

$$F(\mathbf{x}) = 0.25 + \begin{bmatrix} -2 & -1 \end{bmatrix}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix}\mathbf{x}.$$

The gradient at $\mathbf{x}_0$ is

$$\mathbf{g}_0 = \nabla F(\mathbf{x}_0) = \mathbf{A}\mathbf{x}_0 + \mathbf{d} = \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \end{bmatrix}.$$

The first search direction is then

$$\mathbf{p}_0 = -\mathbf{g}_0 = \begin{bmatrix} -10 \\ -5 \end{bmatrix}.$$

To minimize along a line, for a quadratic function, we can use Eq. (9.31):

$$\alpha_0 = -\frac{\mathbf{g}_0^T \mathbf{p}_0}{\mathbf{p}_0^T \mathbf{A} \mathbf{p}_0} = -\frac{\begin{bmatrix} 10 & 5 \end{bmatrix} \begin{bmatrix} -10 \\ -5 \end{bmatrix}}{\begin{bmatrix} -10 & -5 \end{bmatrix} \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -10 \\ -5 \end{bmatrix}} = -\frac{-125}{1300} = 0.0962 .$$

Therefore the first iteration of conjugate gradient will be

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{p}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + 0.0962 \begin{bmatrix} -10 \\ -5 \end{bmatrix} = \begin{bmatrix} 0.038 \\ 0.519 \end{bmatrix} .$$

Now we need to find the second search direction from Eq. (9.60). This requires the gradient at $\mathbf{x}_1$:

$$\mathbf{g}_1 = \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_1} = \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 0.038 \\ 0.519 \end{bmatrix} + \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.577 \\ 1.154 \end{bmatrix} .$$

We can now find β_1:

$$\beta_1 = \frac{\Delta \mathbf{g}_0^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = \frac{\begin{bmatrix} -10.577 & -3.846 \end{bmatrix} \begin{bmatrix} -0.577 \\ 1.154 \end{bmatrix}}{\begin{bmatrix} 10 & 5 \end{bmatrix} \begin{bmatrix} 10 \\ 5 \end{bmatrix}} = \frac{1.665}{125} = 0.0133 ,$$

using the method of Polak and Ribiére (Eq. (9.63)). (The other two methods for computing β_1 will produce the same results for a quadratic function. You may want to try them.) The second search direction is then computed from Eq. (9.60):

$$\mathbf{p}_1 = -\mathbf{g}_1 + \beta_1 \mathbf{p}_0 = \begin{bmatrix} 0.577 \\ -1.154 \end{bmatrix} + 0.0133 \begin{bmatrix} -10 \\ -5 \end{bmatrix} = \begin{bmatrix} 0.444 \\ -1.220 \end{bmatrix} .$$

From Eq. (9.31), the learning rate for the second iteration will be

$$\alpha_1 = -\frac{\begin{bmatrix} -0.577 & 1.154 \end{bmatrix} \begin{bmatrix} 0.444 \\ -1.220 \end{bmatrix}}{\begin{bmatrix} 0.444 & -1.220 \end{bmatrix} \begin{bmatrix} 10 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 0.444 \\ -1.220 \end{bmatrix}} = -\frac{-1.664}{5.758} = 0.2889 .$$

The second step of conjugate gradient is therefore

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} 0.038 \\ 0.519 \end{bmatrix} + 0.2889 \begin{bmatrix} 0.444 \\ -1.220 \end{bmatrix} = \begin{bmatrix} 0.1667 \\ 0.1667 \end{bmatrix}.$$

As expected, the minimum is reached in two iterations. The trajectory is illustrated in Figure P9.9.

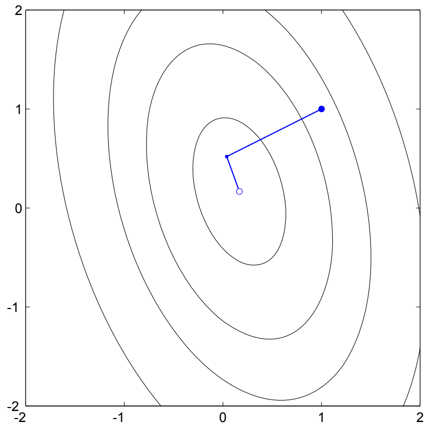

Figure P9.9 Conjugate Gradient Trajectory for Problem P9.7

P9.8 Show that conjugate vectors are independent.

Suppose that we have a set of vectors, $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{n-1}\}$, which are conjugate with respect to the Hessian matrix $\mathbf{A}$. If these vectors are dependent, then, from Eq. (5.4), it must be true that

$$\sum_{j=0}^{n-1} a_j \mathbf{p}_j = \mathbf{0},$$

for some set of constants $a_0, a_1, \ldots, a_{n-1}$, at least one of which is nonzero. If we multiply both sides of this equation by $\mathbf{p}_k^T \mathbf{A}$, we obtain

$$\mathbf{p}_k^T \mathbf{A} \sum_{j=0}^{n-1} a_j \mathbf{p}_j = \sum_{j=0}^{n-1} a_j \mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = a_k \mathbf{p}_k^T \mathbf{A} \mathbf{p}_k = 0,$$

where the second equality comes from the definition of conjugate vectors in Eq. (9.52). If $\mathbf{A}$ is positive definite (a unique strong minimum exists), then $\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k$ must be strictly positive. This implies that a_k must be zero for all k. Therefore conjugate directions must be independent.

Epilogue

In this chapter we have introduced three different optimization algorithms: steepest descent, Newton's method and conjugate gradient. The basis for these algorithms is the Taylor series expansion. Steepest descent is derived by using a first-order expansion, whereas Newton's method and conjugate gradient are designed for second-order (quadratic) functions.

Steepest descent has the advantage that it is very simple, requiring calculation only of the gradient. It is also guaranteed to converge to a stationary point if the learning rate is small enough. The disadvantage of steepest descent is that training times are generally longer than for other algorithms. This is especially true when the eigenvalues of the Hessian matrix, for quadratic functions, have a wide range of magnitudes.

Newton's method is generally much faster than steepest descent. For quadratic functions it will locate a stationary point in one iteration. One disadvantage is that it requires calculation and storage of the Hessian matrix, as well as its inverse. In addition, the convergence properties of Newton's method are quite complex. In Chapter 12 we will introduce a modification of Newton's method that overcomes some of the disadvantages of the standard algorithm.

The conjugate gradient algorithm is something of a compromise between steepest descent and Newton's method. It will locate the minimum of a quadratic function in a finite number of iterations, but it does not require calculation and storage of the Hessian matrix. It is well suited to problems with large numbers of parameters, where it is impractical to compute and store the Hessian.

In later chapters we will apply each of these optimization algorithms to the training of neural networks. In Chapter 10 we will demonstrate how an approximate steepest descent algorithm, Widrow-Hoff learning, can be used to train linear networks. In Chapter 11 we generalize Widrow-Hoff learning to train multilayer networks. In Chapter 12 the conjugate gradient algorithm, and a variation of Newton's method, are used to speed up the training of multilayer networks.

Further Reading

[Batt92] R. Battiti, “First and Second Order Methods for Learning: Between Steepest Descent and Newton’s Method,” *Neural Computation*, Vol. 4, No. 2, pp. 141-166, 1992.

This article reviews the latest developments in unconstrained optimization using first and second derivatives. The techniques discussed are those that are most suitable for neural network applications.

[Brog91] W. L. Brogan, *Modern Control Theory,* 3rd Ed., Englewood Cliffs, NJ: Prentice-Hall, 1991.

This is a well-written book on the subject of linear systems. The first half of the book is devoted to linear algebra. It also has good sections on the solution of linear differential equations and the stability of linear and nonlinear systems. It has many worked problems.

[Gill81] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, New York: Academic Press, 1981.

As the title implies, this text emphasizes the practical implementation of optimization algorithms. It provides motivation for the optimization methods, as well as details of implementation that affect algorithm performance.

[Himm72] D. M. Himmelblau, *Applied Nonlinear Programming*, New York: McGraw-Hill, 1972.

This is a comprehensive text on nonlinear optimization. It covers both constrained and unconstrained optimization problems. The text is very complete, with many examples worked out in detail.

[Scal85] L. E. Scales, *Introduction to Non-Linear Optimization*, New York: Springer-Verlag, 1985.

A very readable text describing the major optimization algorithms, this text emphasizes methods of optimization rather than existence theorems and proofs of convergence. Algorithms are presented with intuitive explanations, along with illustrative figures and examples. Pseudo-code is presented for most algorithms.

Exercises

E9.1 In Problem P9.1 we found the maximum stable learning rate for the steepest descent algorithm when applied to a particular quadratic function. Will the algorithm always diverge when a larger learning rate is used, or are there any conditions for which the algorithm will still converge?

E9.2 We want to find the minimum of the following function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 6 & -2 \\ -2 & 6 \end{bmatrix} \mathbf{x} + \begin{bmatrix} -1 & -1 \end{bmatrix} \mathbf{x} .$$

i. Sketch a contour plot of this function.

ii. Sketch the trajectory of the steepest descent algorithm on the contour plot of part (i), if the initial guess is $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$. Assume a very small learning rate is used.

iii. Perform two iterations of steepest descent with learning rate $\alpha = 0.1$.

iv. What is the maximum stable learning rate?

v. What is the maximum stable learning rate for the initial guess given in part (ii)? (See Exercise E9.1.)

```
» 2 + 2
ans =
     4
```

vi. Write a MATLAB M-file to implement the steepest descent algorithm for this problem, and use it to check your answers to parts (i). through (v).

E9.3 For the quadratic function

$$F(\mathbf{x}) = x_1^2 + 2x_2^2 ,$$

i. Find the minimum of the function along the line

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \alpha \begin{bmatrix} -1 \\ -2 \end{bmatrix} .$$

ii. Verify that the gradient of $F(\mathbf{x})$ at the minimum point from part (i) is orthogonal to the line along which the minimization occurred.

E9.4 For the functions given in Exercise E8.3 perform two iterations of the steepest descent algorithm with linear minimization, starting from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$. Write MATLAB M-files to check your answer.

E9.5 Consider the following function:

$$F(\mathbf{x}) = [1 + (x_1 + x_2 - 5)^2][1 + (3x_1 - 2x_2)^2] .$$

i. Perform one iteration of Newton's method, starting from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 10 & 10 \end{bmatrix}^T$.

ii. Repeat part (i), starting from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 2 & 2 \end{bmatrix}^T$.

iii. Find the minimum of the function, and compare with your results from the previous two parts.

E9.6 Consider the following quadratic function

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 3 & 2 \\ 2 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 4 & 4 \end{bmatrix} \mathbf{x}$$

i. Sketch the contour plot for $F(\mathbf{x})$. Show all work.

ii. Take one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$.

iii. In part (ii), did you reach the minimum of $F(\mathbf{x})$? Explain.

E9.7 Consider the function

$$F(\mathbf{x}) = (x_1 + x_2)^4 + 2(x_2 - 1)^2$$

i. Find the second-order Taylor series approximation of this function about the point $\mathbf{x}_0 = \begin{bmatrix} -1 & 1 \end{bmatrix}^T$.

ii. Is this point a minimum point? Does it satisfy the first and second order conditions?

iii. Perform one iteration of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 0.5 & 0 \end{bmatrix}^T$.

E9.8 Consider the following quadratic function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\begin{bmatrix} 7 & -9 \\ -9 & -17 \end{bmatrix}\mathbf{x} + \begin{bmatrix} 16 & 8 \end{bmatrix}\mathbf{x}$$

i. Sketch the contour plot for this function.

ii. Take one step of Newton's method from the initial guess $\mathbf{x}_0 = \begin{bmatrix} 2 & 2 \end{bmatrix}^T$.

iii. Did you reach the minimum of the function after the Newton step of part (ii)? Explain.

iv. From the initial guess in part ii, trace the path of steepest descent, with very small learning rate, on your contour plot from part (i). Explain how you determined the path. Will steepest descent eventually converge to the same result you found in part (ii)? Explain.

E9.9 Consider the following function:

$$F(\mathbf{x}) = (1 + x_1 + x_2)^2 + \frac{1}{4}x_1^4 .$$

i. Find the quadratic approximation to $F(\mathbf{x})$ about the point $\mathbf{x}_0 = \begin{bmatrix} 2 & 2 \end{bmatrix}^T$.

ii. Sketch the contour plot of the quadratic approximation in part i.

iii. Perform one iteration of Newton's method on the function $F(\mathbf{x})$ from the initial condition $\mathbf{x}_0$ given in part (i). Sketch the path from $\mathbf{x}_0$ to $\mathbf{x}_1$ on your contour plot from part (ii).

iv. Is the $\mathbf{x}_1$ in part iii. a strong minimum of the quadratic approximation? Is it a strong minimum of the original function $F(\mathbf{x})$? Explain.

v. Will Newton's method always converge to a strong minimum of $F(\mathbf{x})$, given enough iterations? Will it always converge to a strong minimum of the quadratic approximation of $F(\mathbf{x})$? Explain your answers in detail.

```
» 2 + 2
ans =
     4
```

E9.10 Recall the function presented in Exercise E8.5. Write MATLAB M-files to implement the steepest descent algorithm and Newton's method for that function. Test the performance of the algorithms for various initial guesses.

E9.11 Repeat Exercise E9.4 using the conjugate gradient algorithm. Use each of the three methods (Eq. (9.61)–Eq. (9.63)) at least once.

E9.12 Prove or disprove the following statement:

If $\mathbf{p}_1$ is conjugate to $\mathbf{p}_2$ and $\mathbf{p}_2$ is conjugate to $\mathbf{p}_3$,
then $\mathbf{p}_1$ is conjugate to $\mathbf{p}_3$.