Conjugate gradient algorithm for training neural networks

1. Introduction

Recall that in the *steepest-descent* neural network training algorithm, consecutive line-search directions are *orthogonal*, such that,

$$\mathbf{g}[\mathbf{w}(t+1)]^T\mathbf{d}(t) = 0 \tag{1}$$

where, $\mathbf{g}[\mathbf{w}(t+1)]$ denotes $\nabla E[\mathbf{w}(t+1)]$, the gradient of the error E with respect to the weights $\mathbf{w}(t+1)$ at step $(t+1)$ of the training algorithm, and $\mathbf{d}(t)$ denotes the line-search direction at step t of the training algorithm. In steepest descent, of course,

$$\mathbf{d}(t) = -\mathbf{g}(t), \tag{2}$$

and,

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta(t)\mathbf{g}(t), \tag{3}$$

where $\eta(t)$ is determined at each step through line minimization.

2. Deriving the conjugate gradient algorithm

A. Introduction

As we have seen, choosing consecutive line-search directions as in equation (2) can lead to oscillatory behavior in the training algorithm, even for simple quadratic error surfaces. This can significantly slow down convergence of the algorithm. To combat this, we should ideally choose consecutive line-search directions that are non-interfering — in other words, consecutive search directions that do not undo the progress of previous search directions. Suppose we have executed a line-search along direction $\mathbf{d}(t)$ at time step t; then the next search direction $\mathbf{d}(t+1)$ should be chosen so that,

$$\mathbf{g}[\mathbf{w}(t+1) + \eta\mathbf{d}(t+1)]^T\mathbf{d}(t) = 0, \ \forall\eta. \tag{4}$$

Condition (4) explicitly preserves progress made in the previous line search (see Figure 7.11 in [1]), and will guide us through our development of the conjugate gradient training algorithm. Below, we closely follow the derivation in [1], but provide somewhat greater detail in certain parts of the derivation.

B. First step

Lemma: Condition (4) implies,

$$\mathbf{d}(t+1)^T\mathbf{H}\mathbf{d}(t) \tag{5}$$

to first order, where $\mathbf{H}$ is the Hessian matrix with elements $\mathbf{H}_{(i,j)}$,

$$\mathbf{H}_{(i,j)} = \frac{\partial^2 E}{\partial\omega_i\partial\omega_j}, \tag{6}$$

evaluated at $\mathbf{w}(t+1)$, where,

$$\mathbf{w} = \left[\omega_1 \ \omega_2 \ \ldots \ \omega_W\right]^T. \tag{7}$$

Proof: First, let us approximate $\mathbf{g}(\mathbf{w})$ by its first-order Taylor approximation about $\mathbf{w}(t+1)$,

$$\mathbf{g}(\mathbf{w})^T \approx \mathbf{g}[\mathbf{w}(t+1)]^T + [\mathbf{w} - \mathbf{w}(t+1)]^T\nabla\{\mathbf{g}[\mathbf{w}(t+1)]\} \tag{8}$$

and evaluate the right-hand side of equation (8) at $\mathbf{w}(t+1) + \eta\mathbf{d}(t+1)$:

$$\mathbf{g}[\mathbf{w}(t+1) + \eta\mathbf{d}(t+1)]^T \approx \mathbf{g}[\mathbf{w}(t+1)]^T + \eta\mathbf{d}(t+1)^T\mathbf{H}. \tag{9}$$

Note from equation (9) that,

$$\mathbf{H} = \nabla\{\nabla E[\mathbf{w}(t+1)]\} = \nabla\{\mathbf{g}[\mathbf{w}(t+1)]\}. \tag{10}$$

Substituting equation (9) into (4),

$$\{\mathbf{g}[\mathbf{w}(t+1)]^T + \eta\mathbf{d}(t+1)^T\mathbf{H}\}\mathbf{d}(t) = 0 \tag{11}$$

$$\mathbf{g}[\mathbf{w}(t+1)]^T\mathbf{d}(t) + \eta\mathbf{d}(t+1)^T\mathbf{H}\mathbf{d}(t) = 0. \tag{12}$$

From (1),

$$\mathbf{g}[\mathbf{w}(t+1)]^T\mathbf{d}(t) = 0, \tag{13}$$

so that equation (12) reduces to,

$$\eta\mathbf{d}(t+1)^T\mathbf{H}\mathbf{d}(t) = 0. \tag{14}$$

Equation (14) implies either $\eta = 0$ and/or $\mathbf{d}(t+1)^T\mathbf{H}\mathbf{d}(t)$. We can dismiss the first of these implications ($\eta = 0$), since a zero learning parameter would mean that we are not changing the weights. Therefore, (5) follows from (4) and the proof is complete. ❑

Note that for quadratic error surfaces, the first-order Taylor approximation in (8) is exact. For non-quadratic error surfaces, it is approximately correct in the locally quadratic neighborhood of the weight vector $\mathbf{w}(t+1)$.

Search directions that meet condition (5) are called $\mathbf{H}$*-orthogonal* or *conjugate with respect to* $\mathbf{H}$. Our overall goal in developing the conjugate gradient algorithm is to construct consecutive search directions $\mathbf{d}(t)$ that meet condition (5) without explicit computation of the large Hessian matrix ($\mathbf{H}$). Note that an algorithm that uses $\mathbf{H}$-orthogonal search directions will converge to the minimum of the error surface (for quadratic or locally quadratic error surfaces) in at most W steps, where W is the number of weights in the neural network.

C. Linear algebra preliminaries

In our derivation of the conjugate gradient algorithm, we will assume a locally quadratic error surface $E(\mathbf{w})$ of the following form:

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{15}$$

where E_0 is a constant scalar, $\mathbf{b}$ is a constant vector, and $\mathbf{H}$ is a constant matrix (i.e. the Hessian). Furthermore, we assume that $\mathbf{H}$ is *symmetric* and *positive-definite*, a property that we will sometimes denote by $\mathbf{H} > 0$.

Definition: A square matrix $\mathbf{H}$ is *positive-definite*, if and only if all its eigenvalues λ_i are greater than zero. If a matrix is positive-definite, then,

$$\mathbf{v}^T\mathbf{H}\mathbf{v} > 0, \ \forall\mathbf{v} \neq 0. \tag{16}$$

Note that for a quadratic error surface, $\mathbf{H}$ is constant and is therefore guaranteed to be positive-definite everywhere. Consider, for example, the following simply quadratic error surface of two weights,

$$E = 20\omega_1^2 + \omega_2^2. \tag{17}$$

For (17), we have previously computed the Hessian to be,

$$\mathbf{H} = \begin{bmatrix} 40 & 0 \\ 0 & 2 \end{bmatrix} \tag{18}$$

with eigenvalues $\lambda_1 = 40$ and $\lambda_2 = 2$. Hence, $\mathbf{H}$ is positive-definite. Note also, that for any error function with continuous partial derivatives, the Hessian is guaranteed to be *symmetric* since,

$$\frac{\partial^2 E}{\partial \omega_i \partial \omega_j} = \frac{\partial^2 E}{\partial \omega_j \partial \omega_i} \tag{19}$$

Theorem: For a positive-definite square matrix $\mathbf{H}$, $\mathbf{H}$-orthogonal vectors $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_k\}$ are *linearly independent*.

Proof: For linear independence, we must show that,

$$\alpha_1 \mathbf{d}_1 + \alpha_2 \mathbf{d}_2 + \ldots + \alpha_k \mathbf{d}_k = \mathbf{0} \tag{20}$$

if and only if $\alpha_i = 0$, $\forall i$. In equation (20), $\mathbf{0}$ denotes a vector with equal dimensionality as $\mathbf{d}_i$ and all zero elements. Let us first, pre-multiply equation (20) by $\mathbf{d}_i^T \mathbf{H}$,

$$\alpha_1 \mathbf{d}_i^T \mathbf{H} \mathbf{d}_1 + \alpha_2 \mathbf{d}_i^T \mathbf{H} \mathbf{d}_2 + \ldots + \alpha_k \mathbf{d}_i^T \mathbf{H} \mathbf{d}_k = \mathbf{d}_i^T \mathbf{H} \mathbf{0} \tag{21}$$

$$\alpha_1 \mathbf{d}_i^T \mathbf{H} \mathbf{d}_1 + \alpha_2 \mathbf{d}_i^T \mathbf{H} \mathbf{d}_2 + \ldots + \alpha_k \mathbf{d}_i^T \mathbf{H} \mathbf{d}_k = 0 \tag{22}$$

Note that since we have assumed that the $\mathbf{d}_i$, $i \in \{1, 2, \ldots, k\}$, vectors are $\mathbf{H}$-orthogonal, i.e.,

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_j = 0, \ \forall i \neq j, \tag{23}$$

all terms in (22) drop out except the $\alpha_i \mathbf{d}_i^T \mathbf{H} \mathbf{d}_i$ term, so that (22) reduces to,

$$\alpha_i \mathbf{d}_i^T \mathbf{H} \mathbf{d}_i = 0 \tag{24}$$

Since $\mathbf{H}$ is positive-definite, however, we know from (16) that,

$$\mathbf{d}_i^T \mathbf{H} \mathbf{d}_i > 0 \tag{25}$$

so that in order for (24) to be true,

$$\alpha_i = 0, \ i \in \{1, 2, \ldots, k\}, \tag{26}$$

and the proof is complete. ❑

From this theorem, it follows directly that in a W-dimensional space, $\mathbf{H}$-orthogonal vectors $\mathbf{d}_i$, $i \in \{1, 2, \ldots, W\}$, form a complete, albeit non-orthogonal, basis set. This means that any vector $\mathbf{v}$ in that space may be expressed as the linear combination of the $\mathbf{d}_i$ vectors, i.e.,

$$\mathbf{v} = \sum_{i=1}^{W} \alpha_i \mathbf{d}_i . \tag{27}$$

D. The setup

This section establishes the foundations of the conjugate gradient algorithm. In our derivation here and subsequent sections, we will assume a locally quadratic error surface of form (15),

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T \mathbf{w} + \frac{1}{2} \mathbf{w}^T \mathbf{H} \mathbf{w} . \tag{28}$$

For now, let us also assume a set of $\mathbf{H}$-orthogonal (or conjugate) vectors $\mathbf{d}_i$, $i \in \{1, 2, \ldots, W\}$. Given these conjugate vectors, let us see how we can move from some initial (nonzero) weight vector $\mathbf{w}_1$ to the minimum $\mathbf{w}^*$ of error surface (28). From (27) we can express the difference between $\mathbf{w}_1$ and $\mathbf{w}^*$ as,

$$\mathbf{w}^* - \mathbf{w}_1 = \sum_{i=1}^{W} \alpha_i \mathbf{d}_i \tag{29}$$

so that,

$$\mathbf{w}^* = \mathbf{w}_1 + \sum_{i=1}^{W} \alpha_i \mathbf{d}_i . \tag{30}$$

Then, if we define,

$$\mathbf{w}_j \equiv \mathbf{w}_1 + \sum_{i=1}^{j-1} \alpha_i \mathbf{d}_i \tag{31}$$

we can rewrite equation (30) recursively as,

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j . \tag{32}$$

Note that equation (32) represents a sequence of steps in weight space from $\mathbf{w}_1$ to $\mathbf{w}^*$. These steps are parallel to the conjugate directions $\mathbf{d}_j$, and the length of each step is controlled by α_j. Equation (32) forms the basis of the conjugate gradient algorithm for training neural networks and is similar to previous training algorithms that we have studied in that it offers a recursive minimization procedure for the weights $\mathbf{w}$. The big difference to previous algorithms is, of course, that $\mathbf{d}_j \neq -\mathbf{g}_j$, as was the case for the gradient and steepest descent algorithms. Also, the step size α_j is not held constant, as is typical in gradient descent.

At this point, several big questions remain unanswered, however. First, how do we construct the $\mathbf{d}_j$ vectors given an error surface? Second, how do we determine the corresponding step sizes α_j such that we are guaranteed convergence in at most W steps on a locally quadratic error surface? Finally, how can we do both without explicit computation of the Hessian $\mathbf{H}$? Below, we answer these questions one by one.

E. Computing correct step sizes

Let us first attempt to find an expression for α_j, assuming that we have a set of W conjugate vectors $\mathbf{d}_i$, $i \in \{1, 2, \ldots, W\}$. To do this, let us pre-multiply equation (29) by $\mathbf{d}_j^T\mathbf{H}$:

$$\mathbf{d}_j^T\mathbf{H}(\mathbf{w}^* - \mathbf{w}_1) = \mathbf{d}_j^T\mathbf{H}\left(\sum_{i=1}^{W} \alpha_i \mathbf{d}_i\right) \tag{33}$$

$$\mathbf{d}_j^T\mathbf{H}(\mathbf{w}^* - \mathbf{w}_1) = \sum_{i=1}^{W} \alpha_i \mathbf{d}_j^T\mathbf{H}\mathbf{d}_i \tag{34}$$

By $\mathbf{H}$-orthogonality, all the terms in the right-hand sum of equation (34) where $i \neq j$ are zero. Thus, equation (34) reduces to,

$$\mathbf{d}_j^T\mathbf{H}(\mathbf{w}^* - \mathbf{w}_1) = \alpha_j \mathbf{d}_j^T\mathbf{H}\mathbf{d}_j . \tag{35}$$

Now, note that since the gradient of (28) is given by,

$$\mathbf{g}(\mathbf{w}) = \mathbf{b} + \mathbf{H}\mathbf{w} , \tag{36}$$

the minimum $\mathbf{w}^*$ can be expressed implicitly as,

$$\mathbf{g}(\mathbf{w}^*) = 0 \tag{37}$$

so that,

$$\mathbf{b} + \mathbf{H}\mathbf{w}^* = 0 , \tag{38}$$

$$\mathbf{H}\mathbf{w}^* = -\mathbf{b} . \tag{39}$$

Consequently, equation (35) further reduces to,

$$\mathbf{d}_j^T(-\mathbf{b} - \mathbf{H}\mathbf{w}_1) = \alpha_j \mathbf{d}_j^T\mathbf{H}\mathbf{d}_j , \tag{40}$$

$$-\mathbf{d}_j^T(\mathbf{b} + \mathbf{H}\mathbf{w}_1) = \alpha_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j . \tag{41}$$

Thus,

$$\alpha_j = \frac{-\mathbf{d}_j^T(\mathbf{b} + \mathbf{H}\mathbf{w}_1)}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{42}$$

Although equation (42) gives an expression for α_j, the right-hand side of (42) is dependent on the initial weights $\mathbf{w}_1$. We can remove this dependence. Starting with equation (31),

$$\mathbf{w}_j = \mathbf{w}_1 + \sum_{i=1}^{j-1} \alpha_i \mathbf{d}_i \tag{43}$$

we (once again) pre-multiply by $\mathbf{d}_j^T \mathbf{H}$,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{w}_j = \mathbf{d}_j^T \mathbf{H} \mathbf{w}_1 + \sum_{i=1}^{j-1} \alpha_i \mathbf{d}_j^T \mathbf{H} \mathbf{d}_i . \tag{44}$$

Since $i \neq j$ for all terms in the sum on the right-hand side of equation (44), the sum is zero by $\mathbf{H}$-orthogonality. Thus, equation (44) reduces to,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{w}_j = \mathbf{d}_j^T \mathbf{H} \mathbf{w}_1 . \tag{45}$$

Substituting (45) into (42) and letting,

$$\mathbf{g}_j = \mathbf{b} + \mathbf{H}\mathbf{w}_j , \tag{46}$$

equation (42) further simplifies to,

$$\alpha_j = \frac{-\mathbf{d}_j^T(\mathbf{b} + \mathbf{H}\mathbf{w}_j)}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{47}$$

$$\alpha_j = \frac{-\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{48}$$

Below, we proof that iteration (32), with step size α_j as given in (48), will converge to $\mathbf{w}^*$ in at most W steps from any nonzero initial weight vector $\mathbf{w}_1$ and a quadratic error surface.

Theorem: Assuming a W-dimensional quadratic error surface,

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T \mathbf{w} + \frac{1}{2} \mathbf{w}^T \mathbf{H} \mathbf{w} , \tag{49}$$

and $\mathbf{H}$-orthogonal vectors $\mathbf{d}_i$, $i \in \{1, 2, \ldots, W\}$, the following weight iteration,

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j , \tag{50}$$

$$\alpha_j = \frac{-\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} , \tag{51}$$

will converge in at most W steps to the minimum $\mathbf{w}^*$ of the error surface $E(\mathbf{w})$ from any nonzero initial weight vector $\mathbf{w}_1$.

Proof: We will proof this theorem by showing that the gradient vector $\mathbf{g}_j$ at the jth iteration is orthogonal to all previous conjugate directions $\mathbf{d}_i$, $i < j$. From this, it follows that after W steps, the components of the gradient along all directions will have been made zero, and consequently, iteration (50) will have converged to the minimum $\mathbf{w}^*$.

Since,

$$\mathbf{g}_j = \mathbf{b} + \mathbf{H}\mathbf{w}_j \tag{52}$$

we can write,

$$\mathbf{g}_{j+1} - \mathbf{g}_j = \mathbf{H}(\mathbf{w}_{j+1} - \mathbf{w}_j) . \tag{53}$$

From (50),

$$(\mathbf{w}_{j+1} - \mathbf{w}_j) = \alpha_j \mathbf{d}_j \tag{54}$$

so that (53) reduces to,

$$\mathbf{g}_{j+1} - \mathbf{g}_j = \alpha_j \mathbf{H}\mathbf{d}_j . \tag{55}$$

Not surprisingly, we now pre-multiply equation (55) by $\mathbf{d}_j^T$,

$$\mathbf{d}_j^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = \alpha_j \mathbf{d}_j^T \mathbf{H}\mathbf{d}_j \tag{56}$$

and substitute (51) for α_j so that,

$$\mathbf{d}_j^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = \left(\frac{-\mathbf{d}_j^T\mathbf{g}_j}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j}\right)\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j \tag{57}$$

$$\mathbf{d}_j^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = -\mathbf{d}_j^T\mathbf{g}_j \tag{58}$$

$$\mathbf{d}_j^T\mathbf{g}_{j+1} - \mathbf{d}_j^T\mathbf{g}_j = -\mathbf{d}_j^T\mathbf{g}_j \tag{59}$$

$$\mathbf{d}_j^T\mathbf{g}_{j+1} = 0 . \tag{60}$$

Therefore, from the result in (60), we know that the gradient at the $(j+1)$ th step is orthogonal to the previous direction $\mathbf{d}_j$. To show that the same is also true for all previous conjugate directions $\mathbf{d}_k$, $k < j$, let us pre-multiply equation (55) by $\mathbf{d}_k^T$,

$$\mathbf{d}_k^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = \alpha_j \mathbf{d}_k^T \mathbf{H}\mathbf{d}_j . \tag{61}$$

Since $k < j$, the right-hand side of equation (61) is zero by $\mathbf{H}$-orthogonality. Thus, (61) reduces to,

$$\mathbf{d}_k^T(\mathbf{g}_{j+1} - \mathbf{g}_j) = 0 \tag{62}$$

$$\mathbf{d}_k^T\mathbf{g}_{j+1} = \mathbf{d}_k^T\mathbf{g}_j \ , \ k < j . \tag{63}$$

Through induction of equations (63) and (60), we can easily show that,

$$\mathbf{d}_k^T\mathbf{g}_j = 0 \ , \ \forall k < j . \tag{64}$$

For example, let $k = j - 1$. From (63),

$$\mathbf{d}_{j-1}^T\mathbf{g}_{j+1} = \mathbf{d}_{j-1}^T\mathbf{g}_j \ , \tag{65}$$

and from (60),

$$\mathbf{d}_{j-1}^T\mathbf{g}_j = 0 \tag{66}$$

so that,

$$\mathbf{d}_{j-1}^T\mathbf{g}_{j+1} = 0 . \tag{67}$$

Now we can let $k = j-2, j-3, \ldots, 1$ and repeat the steps for $k = j - 1$. This completes the proof. ❑

F. Constructing conjugate vectors

So far in our development, we have assumed a set of mutually $\mathbf{H}$-orthogonal vectors $\{\mathbf{d}_j\}$, but have not talked about how to construct these vectors. The theorem below gives one iterative method for doing just that.

Theorem: Let $\mathbf{d}_j$ be defined as follows:

1. Let,

$$\mathbf{d}_1 = -\mathbf{g}_1 \tag{68}$$

2. Let,

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j, \; j \geq 1, \tag{69}$$

where,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{70}$$

The construction of (68), (69) and (70) generates W mutually $\mathbf{H}$-orthogonal vectors on a W-dimensional quadratic error surface, such that,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0, \; \forall i \neq j. \tag{71}$$

Proof: First, we will show that our choice of β_j in (70) implies,

$$\mathbf{d}_{j+1}^T \mathbf{H} \mathbf{d}_j = 0. \tag{72}$$

In other words, we will show that consecutive search directions are $\mathbf{H}$-orthogonal. Pre-multiplying (69) by $\mathbf{d}_j^T \mathbf{H}$,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_{j+1} = -\mathbf{d}_j^T \mathbf{H} \mathbf{g}_{j+1} + \beta_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j, \tag{73}$$

and substituting (70) into (73), we get,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_{j+1} = -\mathbf{d}_j^T \mathbf{H} \mathbf{g}_{j+1} + \left(\frac{\mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \right) \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j \tag{74}$$

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_{j+1} = -\mathbf{d}_j^T \mathbf{H} \mathbf{g}_{j+1} + \mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j = 0. \tag{75}$$

Next, we demonstrate that if,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0 \quad \Rightarrow \quad \mathbf{d}_{j+1}^T \mathbf{H} \mathbf{d}_i = 0, \; \forall i < j, \tag{76}$$

($\Rightarrow$ means “implies”), then,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0, \; \forall i \neq j. \tag{77}$$

This can be shown by induction of (72) and (76); for example,

$$\mathbf{d}_2^T \mathbf{H} \mathbf{d}_1 = 0 \text{ (from (72), by construction)} \tag{78}$$

$$\mathbf{d}_3^T \mathbf{H} \mathbf{d}_1 = 0 \text{ (from (78) and (76), by assumption)} \tag{79}$$

$$\mathbf{d}_3^T \mathbf{H} \mathbf{d}_2 = 0 \text{ (from (72), by construction)} \tag{80}$$

$$\mathbf{d}_4^T \mathbf{H} \mathbf{d}_1 = 0 \text{ (from (79) and (76), by assumption)} \tag{81}$$

$$\mathbf{d}_4^T\mathbf{H}\mathbf{d}_2 = 0 \text{ (from (80) and (76), by assumption)} \tag{82}$$

$$\mathbf{d}_4^T\mathbf{H}\mathbf{d}_3 = 0 \text{ (from (72), by construction), etc.} \tag{83}$$

Thus, to complete the proof, we need to show that (76) is true. First, we transpose and post-multiply (69) by $\mathbf{H}\mathbf{d}_i$, $i < j$,

$$\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_i = -\mathbf{g}_{j+1}^T\mathbf{H}\mathbf{d}_i + \beta_j\mathbf{d}_j^T\mathbf{H}\mathbf{d}_i \tag{84}$$

Now, by assumption (76), $\mathbf{d}_j^T\mathbf{H}\mathbf{d}_i = 0$. Hence,

$$\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_i = -\mathbf{g}_{j+1}^T\mathbf{H}\mathbf{d}_i . \tag{85}$$

From (32), we have that,

$$\mathbf{w}_{j+1} - \mathbf{w}_j = \alpha_j\mathbf{d}_j , \tag{86}$$

$$\mathbf{H}(\mathbf{w}_{j+1} - \mathbf{w}_j) = \alpha_j\mathbf{H}\mathbf{d}_j . \tag{87}$$

Since,

$$\mathbf{g}_j = \mathbf{H}\mathbf{w}_j + \mathbf{b} , \tag{88}$$

$$\mathbf{H}(\mathbf{w}_{j+1} - \mathbf{w}_j) = \mathbf{g}_{j+1} - \mathbf{g}_j . \tag{89}$$

Combining equations (87) and (89),

$$\alpha_j\mathbf{H}\mathbf{d}_j = \mathbf{g}_{j+1} - \mathbf{g}_j \tag{90}$$

$$\mathbf{H}\mathbf{d}_j = \frac{1}{\alpha_j}(\mathbf{g}_{j+1} - \mathbf{g}_j) . \tag{91}$$

We substitute equation (91) into the right-hand side of equation (85) [switching the index from j to i in equation (91)], so that,

$$\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_i = -\frac{1}{\alpha_i}\mathbf{g}_{j+1}^T(\mathbf{g}_{i+1} - \mathbf{g}_i) , \tag{92}$$

$$\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_i = -\frac{1}{\alpha_i}(\mathbf{g}_{j+1}^T\mathbf{g}_{i+1} - \mathbf{g}_{j+1}^T\mathbf{g}_i) , \tag{93}$$

$$\mathbf{d}_{j+1}^T\mathbf{H}\mathbf{d}_i = -\frac{1}{\alpha_i}(\mathbf{g}_{i+1}^T\mathbf{g}_{j+1} - \mathbf{g}_i^T\mathbf{g}_{j+1}) . \tag{94}$$

Now, we will show that the right-hand side of equation (94) is zero, by showing that,

$$\mathbf{g}_k^T\mathbf{g}_j = 0 , \ \forall k < j . \tag{95}$$

This will complete the proof. From (68) and (69), we see that $\mathbf{d}_k$, $k \geq 1$, is a linear combination of all previous gradient directions,

$$\mathbf{d}_k = -\mathbf{g}_k + \sum_{l=1}^{k-1} \gamma_l\mathbf{g}_l , \tag{96}$$

where the γ_l are scalar coefficients. For example,

$$\mathbf{d}_1 = -\mathbf{g}_1 , \tag{97}$$

$$\mathbf{d}_2 = -\mathbf{g}_2 + \beta_1 \mathbf{d}_1 = -\mathbf{g}_2 - \beta_1 \mathbf{g}_1 , \tag{98}$$

$$\mathbf{d}_3 = -\mathbf{g}_3 + \beta_2 \mathbf{d}_2 = -\mathbf{g}_3 - \beta_2 \mathbf{g}_2 - \beta_2 \beta_1 \mathbf{g}_1 . \tag{99}$$

We have already shown,

$$\mathbf{d}_k^T \mathbf{g}_j = 0 , \ \forall k < j , \tag{100}$$

so that if we transpose and post-multiply equation (96) by $\mathbf{g}_j$, $j > k$,

$$\mathbf{d}_k^T \mathbf{g}_j = -\mathbf{g}_k^T \mathbf{g}_j + \sum_{l=1}^{k-1} \gamma_l \mathbf{g}_l^T \mathbf{g}_k , \tag{101}$$

$$0 = -\mathbf{g}_k^T \mathbf{g}_j + \sum_{l=1}^{k-1} \gamma_l \mathbf{g}_l^T \mathbf{g}_k \ \text{[from (100)]}, \tag{102}$$

$$\mathbf{g}_k^T \mathbf{g}_j = \sum_{l=1}^{k-1} \gamma_l \mathbf{g}_l^T \mathbf{g}_k . \tag{103}$$

Since $\mathbf{d}_1 = -\mathbf{g}_1$,

$$\mathbf{d}_1^T \mathbf{g}_j = -\mathbf{g}_1^T \mathbf{g}_j = 0 , \ j > 1 , \ \text{[from (100)]}, \tag{104}$$

$$\mathbf{g}_1^T \mathbf{g}_j = 0 , \ j > 1 . \tag{105}$$

By induction of equations (103) and (105), we can now show that (95) is true; for example,

$$\mathbf{g}_1^T \mathbf{g}_2 = 0 \ \text{[from (105)]} \tag{106}$$

$$\mathbf{g}_2^T \mathbf{g}_3 = \gamma_1 \mathbf{g}_1^T \mathbf{g}_3 = 0 \ \text{[from (103) and (105)]} \tag{107}$$

$$\mathbf{g}_1^T \mathbf{g}_4 = 0 \ \text{[from (105)]} \tag{108}$$

$$\mathbf{g}_2^T \mathbf{g}_4 = \gamma_1 \mathbf{g}_1^T \mathbf{g}_4 = 0 \ \text{[from (103) and (105)]} \tag{109}$$

$$\mathbf{g}_3^T \mathbf{g}_4 = \gamma_1 \mathbf{g}_1^T \mathbf{g}_4 + \gamma_2 \mathbf{g}_2^T \mathbf{g}_4 = 0 \ \text{[from (103) and (109)]}, \text{ etc.} \tag{110}$$

Thus,

$$\mathbf{g}_k^T \mathbf{g}_j = 0 , \ \forall k < j , \tag{111}$$

so that,

$$\mathbf{d}_{j+1}^T \mathbf{H} \mathbf{d}_i = -\frac{1}{\alpha_i} (\mathbf{g}_{i+1}^T \mathbf{g}_{j+1} - \mathbf{g}_i^T \mathbf{g}_{j+1}) = 0 , \ i < j . \tag{112}$$

Recall that equation (112) presupposes $\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0$, so that we have shown equation (76) to be true; i.e.,

$$\mathbf{d}_{j+1}^T \mathbf{H} \mathbf{d}_i = 0 \ \text{when} \ \mathbf{d}_j^T \mathbf{H} \mathbf{d}_i = 0 , \ \forall i < j . \tag{113}$$

Thus, by induction of equations (72) and (76), the construction in (68), (69) and (70) generates mutually $\mathbf{H}$-orthogonal vectors on a quadratic error surface, such that,

$$\mathbf{d}_j^T \mathbf{H} \mathbf{d}_i , \ \forall i \neq j , \tag{114}$$

and the proof is complete. ❑

G. Computing β_j without the Hessian

We now have the following algorithm for training the weights of a neural network:

1. Let,

$$\mathbf{d}_1 = -\mathbf{g}_1 , \tag{115}$$

where $\mathbf{w}_1$ is a random initial weight vector.

2. For $j \geq 1$ let,

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha_j \mathbf{d}_j \tag{116}$$

where,

$$\alpha_j = \frac{-\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{117}$$

3. Let,

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j , \tag{118}$$

where,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T \mathbf{H} \mathbf{d}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{119}$$

4. Iterate steps 2 and 3.

Ideally, we would like to compute both α_j and β_j without explicit computation of $\mathbf{H}$. Let us look at β_j first. Substituting equation (91),

$$\mathbf{H} \mathbf{d}_j = \frac{1}{\alpha_j} (\mathbf{g}_{j+1} - \mathbf{g}_j) \tag{120}$$

into equation (119), we get,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T (\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{d}_j^T (\mathbf{g}_{j+1} - \mathbf{g}_j)} \tag{121}$$

which is known as the *Hestenes-Stiefel* expression for β_j. Expression (121) can be further simplified. From equation (118),

$$\mathbf{d}_j = -\mathbf{g}_j + \beta_{j-1} \mathbf{d}_{j-1} . \tag{122}$$

Then, transposing and post-multiplying equation (122) by $\mathbf{g}_j$, we get,

$$\mathbf{d}_j^T \mathbf{g}_j = -\mathbf{g}_j^T \mathbf{g}_j + \beta_{j-1} \mathbf{d}_{j-1}^T \mathbf{g}_j . \tag{123}$$

Since

$$\mathbf{d}_k^T \mathbf{g}_j = 0 , \ \forall k < j , \tag{124}$$

equation (123) reduces to,

$$\mathbf{d}_j^T \mathbf{g}_j = -\mathbf{g}_j^T \mathbf{g}_j , \tag{125}$$

$$\mathbf{g}_j^T \mathbf{g}_j = -\mathbf{d}_j^T \mathbf{g}_j . \tag{126}$$

Consequently, equation (121) reduces to,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T(\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T\mathbf{g}_j} \tag{127}$$

which is known as the *Polak-Ribiere* expression for β_j. Finally since,

$$\mathbf{g}_k^T\mathbf{g}_j = 0\,,\ \forall k < j\,, \tag{128}$$

equation (127) further reduces to,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T\mathbf{g}_{j+1}}{\mathbf{g}_j^T\mathbf{g}_j} \tag{129}$$

which is known as the *Fletcher-Reeves* expression for β_j. Note that none of the three β_j expressions (equations (121), (127) and (129), respectively) requires computation of the local Hessian $\mathbf{H}$. Also, note that on a purely quadratic error surface, the three expressions above are identical. Since a neural network error function is typically only approximately quadratic, however, the different expressions for β_j will give slightly different results. According to [1], the Polak-Ribiere form in (127) usually gives slightly better results in practice than the other two expressions. With (127), when the algorithm is making poor progress, β_j will be approximately zero, so that the search direction will be reset to the negative gradient, effectively re-initializing or restarting the conjugate gradient algorithm.

H. Computing α_j without the Hessian

As we will show below, rather than compute α_j (the step size along each search direction) through equation (51),

$$\alpha_j = \frac{-\mathbf{d}_j^T\mathbf{g}_j}{\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j}\,\mathrm{p} \tag{130}$$

we can instead determine α_j through a line minimization along $\mathbf{d}_j$. We will show below that for a quadratic error surface, this line minimization is equivalent to computing α_j explicitly. For a quadratic error surface,

$$E(\mathbf{w}) = E_0 + \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{131}$$

we have that,

$$E(\mathbf{w}_j + \alpha_j\mathbf{d}_j) = E_0 + \mathbf{b}^T(\mathbf{w}_j + \alpha_j\mathbf{d}_j) + \frac{1}{2}(\mathbf{w}_j + \alpha_j\mathbf{d}_j)^T\mathbf{H}(\mathbf{w}_j + \alpha_j\mathbf{d}_j)\,. \tag{132}$$

We now differentiate equation (132) with respect to α_j and equate to zero:

$$\frac{\partial E(\mathbf{w}_j + \alpha_j\mathbf{d}_j)}{\partial\alpha_j} = \mathbf{b}^T\mathbf{d}_j + \frac{1}{2}\mathbf{d}_j^T\mathbf{H}(\mathbf{w}_j + \alpha_j\mathbf{d}_j) + \frac{1}{2}(\mathbf{w}_j + \alpha_j\mathbf{d}_j)^T\mathbf{H}\mathbf{d}_j = 0 \tag{133}$$

Since $\mathbf{H}$ is symmetric,

$$\mathbf{d}_j^T\mathbf{H}(\mathbf{w}_j + \alpha_j\mathbf{d}_j) = (\mathbf{w}_j + \alpha_j\mathbf{d}_j)^T\mathbf{H}\mathbf{d}_j \tag{134}$$

equation (133) reduces to,

$$\mathbf{b}^T\mathbf{d}_j + \mathbf{d}_j^T\mathbf{H}(\mathbf{w}_j + \alpha_j\mathbf{d}_j) = 0 \tag{135}$$

$$\mathbf{d}_j^T\mathbf{b} + \mathbf{d}_j^T\mathbf{H}\mathbf{w}_j + \alpha_j\mathbf{d}_j^T\mathbf{H}\mathbf{d}_j = 0 \tag{136}$$

$$\alpha_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j = -\mathbf{d}_j^T (\mathbf{b} + \mathbf{H}\mathbf{w}_j) \tag{137}$$

Since,

$$\mathbf{g}_j = \mathbf{b} + \mathbf{H}\mathbf{w}_j , \tag{138}$$

equation (137) reduces to,

$$\alpha_j \mathbf{d}_j^T \mathbf{H} \mathbf{d}_j = -\mathbf{d}_j^T \mathbf{g}_j \tag{139}$$

$$\alpha_j = \frac{-\mathbf{d}_j^T \mathbf{g}_j}{\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j} \tag{140}$$

Note that equation (140) is equivalent to equation (130), which was derived independently from theoretical considerations. Therefore, we have shown that we may safely replace explicit computation of α_j with a line minimization along $\mathbf{d}_j$.

3. The conjugate gradient algorithm

We have now arrived at the complete *conjugate gradient algorithm*, which we summarize below:

1. Choose an initial weight vector $\mathbf{w}_1$ (e.g. vector of small, random weights).

2. Evaluate $\mathbf{g}_1 = \nabla E[\mathbf{w}_1]$ and let,

$$\mathbf{d}_1 = -\mathbf{g}_1 . \tag{141}$$

3. At step j, $j \geq 1$, perform a line minimization along $\mathbf{d}_j$ such that,

$$E(\mathbf{w}_j + \alpha^* \mathbf{d}_j) \leq E(\mathbf{w}_j + \alpha \mathbf{d}_j) , \ \forall \alpha . \tag{142}$$

4. Let,

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \alpha^* \mathbf{d}_j \tag{143}$$

5. Evaluate $\mathbf{g}_{j+1}$.

6. Let,

$$\mathbf{d}_{j+1} = -\mathbf{g}_{j+1} + \beta_j \mathbf{d}_j \tag{144}$$

where,

$$\beta_j = \frac{\mathbf{g}_{j+1}^T (\mathbf{g}_{j+1} - \mathbf{g}_j)}{\mathbf{g}_j^T \mathbf{g}_j} . \tag{145}$$

7. Let $j = j + 1$ and go to step 3.

Note that at no time does the above algorithm require explicit computation of the Hessian $\mathbf{H}$. In fact, the conjugate gradient algorithm requires very little additional computation compared to the steepest descent algorithm. It does, however, exhibit significantly faster convergence properties. In fact, on a W-dimensional quadratic error surface, the conjugate gradient algorithm is guaranteed to converge to the minimum in W steps; no such guarantees exist for the steepest descent algorithm. The main difference between the two algorithms is that the conjugate gradient algorithm exploits a reasonable assumption about the error surface (i.e. locally quadratic) to achieve faster convergence.

A neural network error surface will in general, of course, not be globally quadratic. Therefore, in practice, more than W steps will usually be required before convergence to a local minimum. Consequently, in applying the conjugate gradient algorithm to neural network training, we typically reset the algorithm every W steps; that is,

we reset the search direction to the negative gradient [equation (141)]. Finally we note that throughout our development, we have assumed that the Hessian $\mathbf{H}$ is positive definite. When $\mathbf{H}$ is not positive definite, the line minimization in step 3 of the algorithm prevents the algorithm from making negative progress (i.e. increasing the error of the neural network).

4. Examples

In this section, we will examine the conjugate gradient algorithm for some simple examples, including (1) a simple quadratic error surface, (2) a simple non-quadratic error surface, and (3) a simple neural network training example. We will see that the conjugate gradient algorithm significantly outperforms both the steepest descent and gradient descent algorithms in speed of convergence.

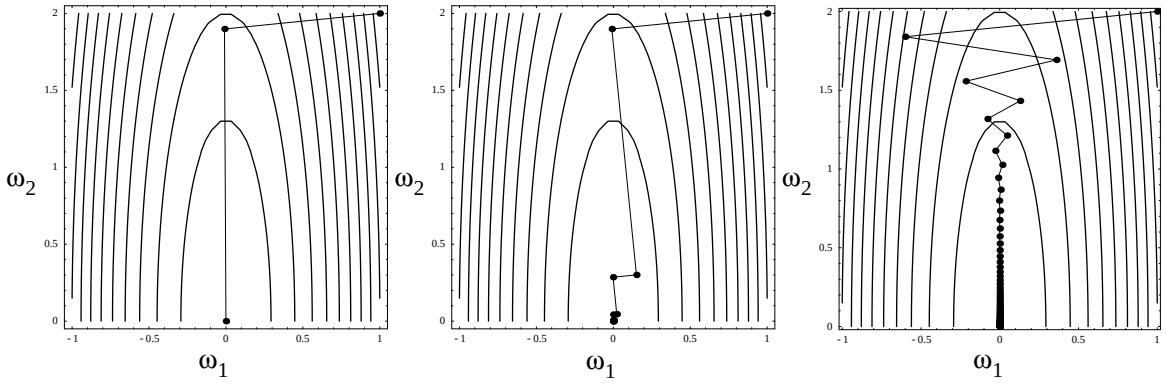
A. Quadratic example

Here we consider the simple quadratic error surface,

$$E = 20\omega_1^2 + \omega_2^2 . \tag{146}$$

For this error surface and from initial weights $(\omega_1, \omega_2) = (1, 2)$, Figure 1 below plots the trajectory in weight space of (1) the conjugate gradient algorithm, (2) the steepest descent algorithm, and (3) the gradient descent algorithm with a learning rate of $\eta = 0.04$. Each trajectory is superimposed on top of a contour plot of E and is stopped when $\sqrt{E} < 10^{-6}$. Note that the minimum for E occurs at $(\omega_1, \omega_2) = (0, 0)$.

From Figure 1, we make several observations. First, as expected, the conjugate gradient algorithm does indeed converge in two steps for a two-dimensional quadratic error surface. By contrast, the steepest descent algorithm requires 15 steps to converge to an error $\sqrt{E} < 10^{-6}$, even though the two algorithms require approximately the same amount of computation per step. The gradient descent algorithm comes in a distant third at 175 steps to convergence for a learning rate $\eta = 0.04$, and this does not count the amount of computation performed in searching for the near-optimal value of the learning rate η. In fact, one of the nicest features of the conjugate gradient algorithm is that it requires *no* hand-selection of any learning parameter, momentum term, etc.



conjugate gradient
2 steps to convergence

steepest descent
15 steps to convergence

gradient descent ($\eta = 0.04$)
175 steps to convergence

Figure 1

A legitimate complaint about Figure 1 is that the conjugate gradient algorithm is specifically designed for a quadratic error surface, so that its superior performance in this case is not surprising — in other words, the skeptical reader might easily conclude that the comparison in Figure 1 is rigged. Therefore, we consider a decidedly non-quadratic error surface in the following section.

B. Non-quadratic example

Here we consider the simple non-quadratic error surface,

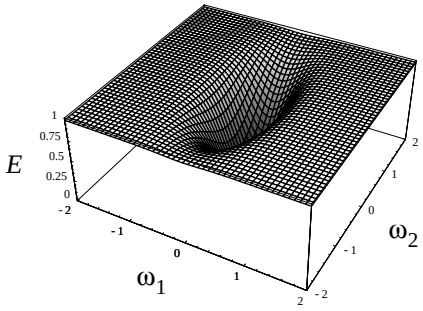


Figure 2

$$E = 1 - \exp(-5\omega_1^2 - \omega_2^2), \tag{147}$$

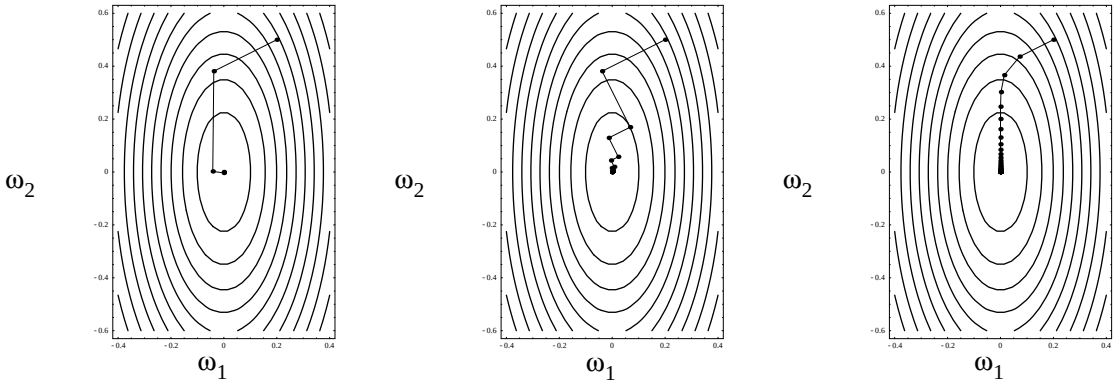
which is plotted in Figure 2. This error surface is characteristic of some regions in weight space for neural network training problems in that for certain weights, the gradient is close to zero.

For this error surface and from initial weights $(\omega_1, \omega_2) = (0.2, 0.5)$, Figure 3 below plots the trajectory in weight space of (1) the conjugate gradient algorithm, (2) the steepest descent algorithm, and (3) the gradient descent algorithm with a learning rate of $\eta = 0.1$. Each trajectory is superimposed on top of a contour plot of E and is stopped when $\sqrt{E} < 10^{-6}$. Note that the minimum for E occurs at $(\omega_1, \omega_2) = (0, 0)$.

From Figure 3, we make several observations. First, the conjugate gradient algorithm no longer converges in just two steps (for a two-dimensional surface), since E in (147) is not quadratic; it does, however, converge in four short steps to $\sqrt{E} < 10^{-6}$. By contrast, the steepest descent algorithm requires 25 steps to converge to an error $\sqrt{E} < 10^{-6}$, and the gradient descent algorithm comes in a distant third at 60 steps to convergence for a learning rate $\eta = 0.1$; once again, the 60 steps for gradient descent does not account for the computation performed in searching for the near-optimal value of the learning rate η.

The observant reader may still be skeptical. Why, you ask, did we choose $(0.2, 0.5)$ as the initial weight vector for this problem? Caught once again! It turns out that these particular initial weights fall just inside the region of the weight space where the Hessian $\mathbf{H}$ is positive definite. The region in Figure 4 encircled by the heavy line corresponds to that region in weight space where $\mathbf{H} > 0$. Note that $(0.2, 0.5)$ falls just inside that region.

So how does the conjugate gradient algorithm fare when $\mathbf{H}$ is not positive definite? Figure 5 answers this question. In Figure 5, we initialize the weights to $(\omega_1, \omega_2) = (1, 2)$, outside the positive-definite region, and once again track the conjugate gradient, steepest descent and gradient descent (with $\eta = 0.1$) algorithms. Note that once again, the conjugate gradient algorithm clearly outperforms its two competitors. In fact, the



conjugate gradient
4 steps to convergence

steepest descent
25 steps to convergence

gradient descent ($\eta = 0.1$)
60 steps to convergence

Figure 3

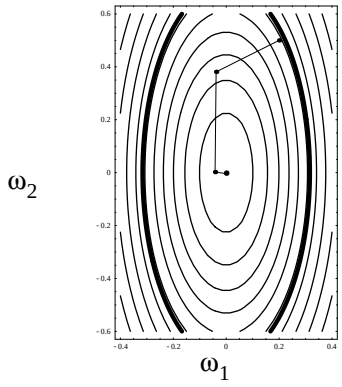
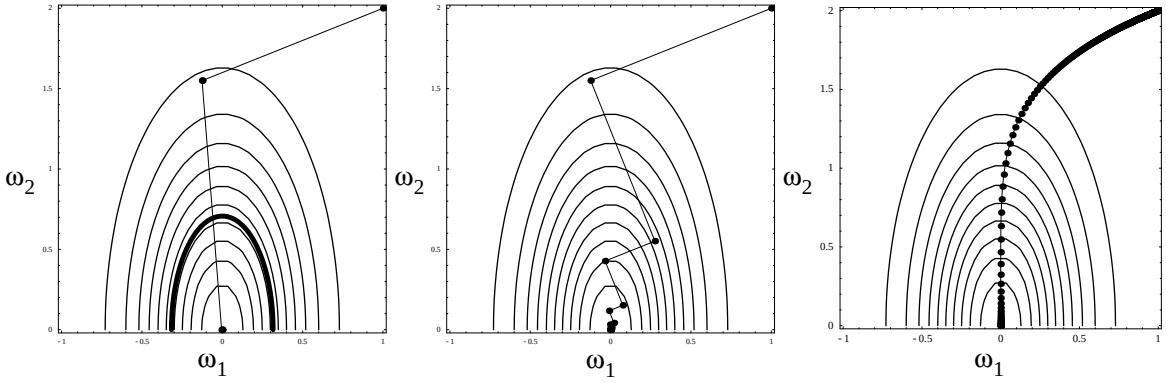


Figure 4



conjugate gradient
5 steps to convergence

steepest descent
24 steps to convergence

gradient descent ($\eta = 0.1$)
940 steps to convergence

Figure 5

gradient descent algorithm degenerates to an awful 940 steps to convergence —this poor performance is typical of gradient descent in flat regions of the error surface.

C. Neural-network training example

Finally, we look at a simple neural network training example. Here we are interested in approximating,

$$y = \frac{1}{2} + \frac{1}{2}\sin(2\pi x),\ 0 \le x \le 1,\ \text{(see Figure 6)} \tag{148}$$

with a single-layer, three-hidden-unit neural network[1] (see Figure 6) and symmetric sigmoidal activation functions,

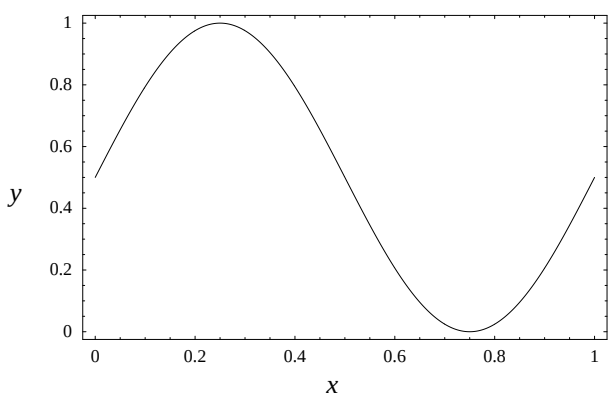
$$\gamma(u) = \frac{1}{1 + e^{-u}} - \frac{1}{2}. \tag{149}$$

Our training data for this problem consists of $N = 500$ evenly spaced (x, y) points sampled from $x \in [0, 1]$.

Figure 7 plots,

$$\log_{10}\sqrt{\frac{E}{N}} \tag{150}$$

1. *Due to an oversight, no connections from the bias unit to the output unit was included in this network. Thus, there are a total of nine trainable weights in this neural network.*



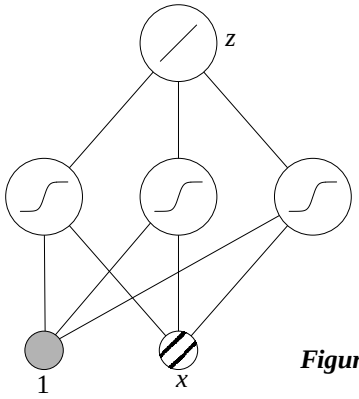


Figure 6

as a function of the number of steps (or epochs) of the conjugate training algorithm for the first 200 steps. By comparison, Figure 8 illustrates that steepest descent does not come close to approaching the conjugate gradient algorithm's performance, even after 500 steps. Finally, we note that gradient descent (with $\eta = 0.3$) requires 10,000 epochs to reach approximately the same error as the conjugate gradient algorithm does in 200 epochs, as shown in Figure 9. Once again, the conjugate gradient algorithm converges in many fewer steps than either of its two competitors.

D. Postscript

Let us look at two more issues with regard to the neural network training example of the previous section. First, Figure 10 plots the neural network's progress in approximating y as the number of epochs (steps) increases from 1 to 200. Note how the conjugate gradient algorithm slowly begins to bend an initially linear approximation to ultimately generate a very good fit of the sine curve.

Second, it is interesting to examine how this neural network, with sigmoidal activation functions centered about the origin [see equation (149)] is able to approximate the function y (with non-zero average value) without a bias connection to the output unit z. Let us denote z_i, $i \in \{1, 2, 3\}$, as the net contribution from hidden unit i to the output z, so that,

$$z = z_1 + z_2 + z_3 . \tag{151}$$

In other words,

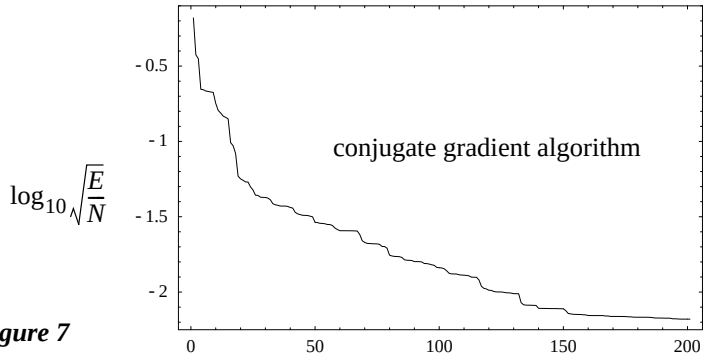
$$z_i = \omega_i h_i , i \in \{1, 2, 3\} , \tag{152}$$

where ω_i denotes the weight from the ith hidden unit to the output, and h_i denotes the output of hidden unit i. Figure 11 (top) plots each of the z_i, which are easily recognized as different parts of the symmetric sigmoid function in equation (149). To understand how they combine to form an approximation of y, we separately plot $z_1 + z_2$ and z_3 in Figure 11 (bottom). Note that z_3 is primarily responsible for the overall shape of the z function, while $z_1 + z_2$ together "pull" the ends of z to conform to the familiar sine curve shape.

5. Conclusion

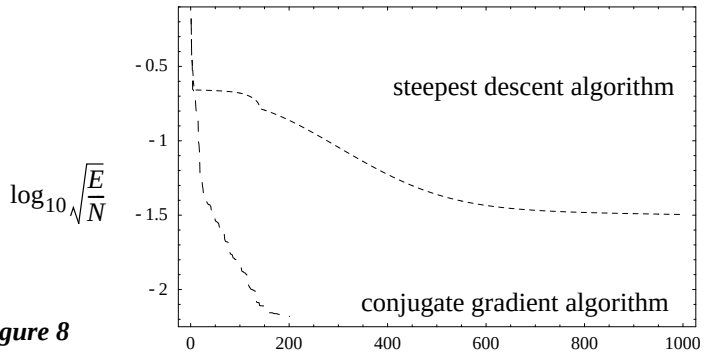
We have derived the conjugate gradient algorithm and illustrated its properties on some simple examples. We have shown that it outperforms both the gradient descent as well as the steepest descent algorithms in terms of the number of computations required to reach convergence. This improvement has been achieved primarily by accounting for the local properties of error surfaces, allowing us to employ gradient information in a more intelligent manner; furthermore, the conjugate gradient algorithm makes unnecessary the often painful trial-and-error selection of tunable learning parameters.

[1] C. M. Bishop, "Chapter 7: Parameter Optimization Algorithms," *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.



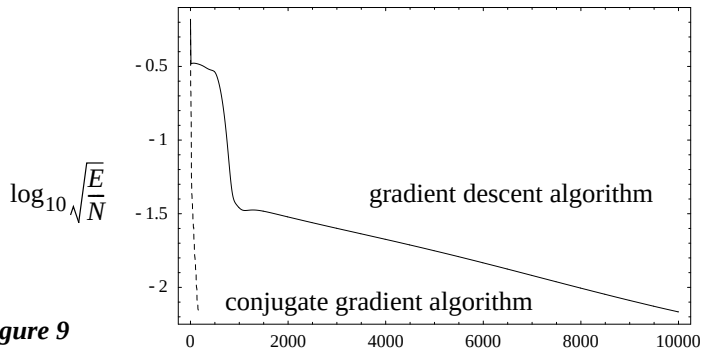
$\log_{10}\sqrt{\frac{E}{N}}$

Figure 7



$\log_{10}\sqrt{\frac{E}{N}}$

Figure 8



$\log_{10}\sqrt{\frac{E}{N}}$

Figure 9

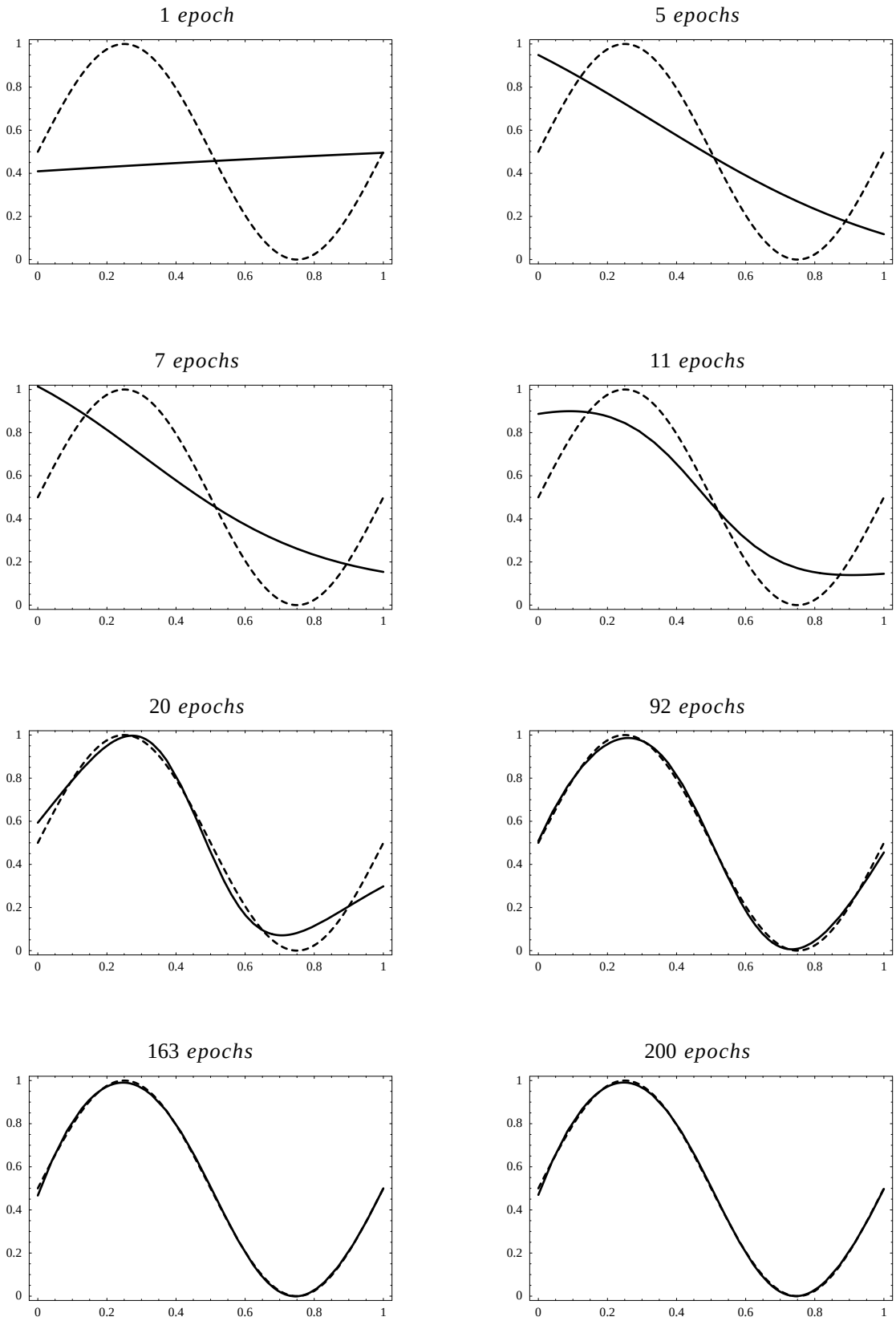

1 epoch
5 epochs
7 epochs
11 epochs
20 epochs
92 epochs
163 epochs
200 epochs

Figure 10

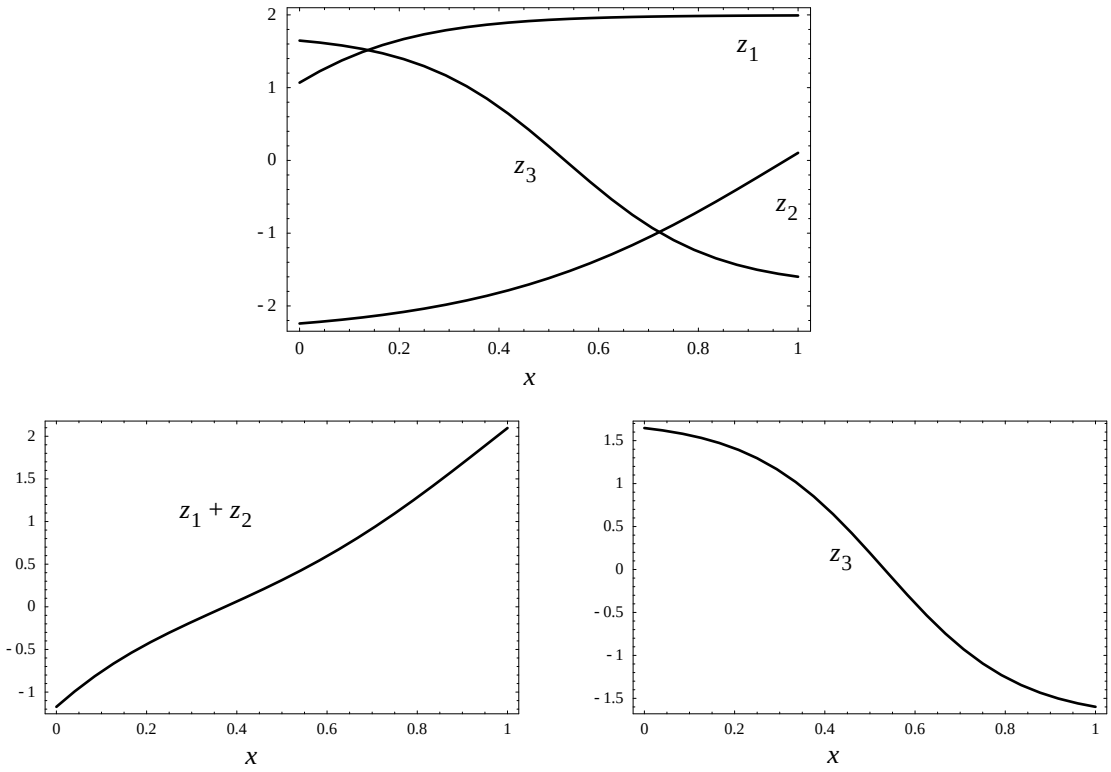

2
1
0
- 1
- 2
0
0.2
0.4
0.6
0.8
1
z_1
z_3
z_2
x
2
1.5
1
0.5
0
- 0.5
- 1
$z_1 + z_2$
x
1.5
1
0.5
0
- 0.5
- 1
- 1.5
z_3
x

Figure 11