Computing Second Derivatives in Feed-Forward Networks: a Review

Wray L. Buntine
RIACS & NASA Ames Research Center
Mail Stop 269–2
Moffet Field, CA 94035–1000, USA
`wray@kronos.arc.nasa.gov`

Andreas S. Weigend*
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304, USA
`weigend@cs.colorado.edu`

Abstract. The calculation of second derivatives is required by recent training and analyses techniques of connectionist networks, such as the elimination of superfluous weights, and the estimation of confidence intervals both for weights and network outputs. We here review and develop exact and approximate algorithms for calculating second derivatives. For networks with $|w|$ weights, simply writing the full matrix of second derivatives requires $O(|w|^2)$ operations. For networks of radial basis units or sigmoid units, exact calculation of the necessary intermediate terms requires of the order of $2h + 2$ backward/forward-propagation passes where h is the number of hidden units in the network. We also review and compare three approximations (ignoring some components of the second derivative, numerical differentiation, and scoring). Our algorithms apply to arbitrary activation functions, networks, and error functions (for instance, with connections that skip layers, or radial basis functions, or cross-entropy error and Softmax units, etc.).

To appear in: *IEEE Transactions on Neural Networks*
(Submitted May 1991. Revised February 1993.)

*Address after August 1993:
Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, USA.

1 Introduction

In the last decade, error backpropagation (Rumelhart *et al.* [RHW86]) has emerged as the most popular training method for connectionist neural networks. Several of the more recent variations of backpropagation require second order derivatives in addition to the first derivatives calculated during standard error backpropagation. These second derivatives are typically derivatives either of the network output or of the error function (also called cost function) with respect to the weights. A matrix of second order derivatives is commonly called a Hessian.

Second derivatives can be calculated exactly, calculated using approximations ignoring certain terms, or calculated using numerical differentiation[1]. We here review these calculations.

Second derivatives are important in several different contexts:

- **Within learning: Approximate second-order methods.** Simple backpropagation is a first order gradient descent method. Learning speed can be improved if information from second derivatives is also used, for instance in a Newton-Raphson type framework [PFTV86, Section 9.6]. Since the second derivatives have to be computed for each weight update, the speed of the computation is here crucial.
  For large networks, calculation of the full Hessian was considered prohibitive. In Section 4.1 we review several approximations: Becker and Le Cun [BL88] suggest a simple diagonal approximation to the Hessian. El-Jaroudi and Makhoul [EJM90] make a block matrix approximation. Fahlman [Fah88] uses in the Quickprop algorithm a simple diagonal approximation and also uses numerical differentiation from the error derivatives of the previous cycle to approximate the diagonal terms of the Hessian (which we discuss in Section 4.3).

- **Within learning: Indirect second-order methods.** There is another class of second order optimization algorithms that do not require direct calculation of the Hessian (or any approximation) because they operate in an iterative manner. The conjugate gradient and related algorithms [PFTV86, Section 10.6] are generally considered the most powerful all-purpose minimization algorithms (although it is not clear whether this scales to very large networks). Here second derivatives are used during line search, so the full Hessian is not required, just the product of the Hessian and a given vector. Møller [Mo93b, Mo93a] and Pearlmutter [Pea93] have both independently suggest numerical differentiation and exact calculation to compute this product. The calculation can also be used iteratively in the power method[2] [GV89] to efficiently approximate the principle eigenvectors of the Hessian. The principle eigenvectors are used by Le Cun, Simard and Pearlmutter [LSP93] to speed up gradient descent.

- **Within learning: Analysis.** It is well known that the speed of training in least mean square algorithms is related to the ratio of the largest to the smallest eigenvalues. A good description is given by Jacob [Jac88], and further analysis is presented by Le Cun, Kanter and Solla [LKS91]. This ratio is called the condition number and is also associated with the accuracy to which the minimum can be calculated. The condition number can be approximated by approximating the largest and smallest eigenvalues with the power method.

- **After learning: Network pruning.** Second derivatives are also used in a post-training phase. Because these post-training methods do not require second derivatives in each training

[1] Numerical differentiation interprets the definition of a derivative numerically by

$$\frac{\partial E(x)}{\partial x} \approx \frac{1}{\triangle x}\left(E(x+\triangle x) - E(x)\right) .$$

[2] From a random initial assignment v_0, compute $u_{i+1} = H \cdot v_i$ then normalize u_{i+1} to a unit vector v_{i+1}. Iterating causes the eigenvectors with smaller eigenvalues to disappear.

iteration, the efficiency of their computation is less crucial here than in the previous two cases. For example, Le Cun, Denker and Solla [LDS90] ("Optimal Brain Damage") use the Hessian of the error (calculated with the Becker and Le Cun approximation referred to above) to simplify the network by pruning weights in order to achieve good generalization performance. Hassibi and Stork [HS93] ("Optimal Brain Surgery") apply the Sherman-Morrison-Woodbury formula for iterative matrix inversion [GV89, PFTV86] to find the inverse of an approximate Hessian. In Section 4.2 of this paper, we suggest a stronger justification for their approximation.

In Bayesian and some Minimum Descripton Length methods, second derivatives are related to quantities such as the posterior variance of the network weights, and also to the description length of a set of weights used in evaluating the quality of the set of weights. Examples are the estimation of the generalization error of the network and of the precision of the network outputs (i.e., confidence intervals or error bars), as well as the comparison of different networks trained on the same data, see Buntine and Weigend [BW91] and MacKay [Mac92]. The network pruning strategy given by Buntine and Weigend [BW91] has the advantage over Hassibi and Stork in that it does not require calculation of the inverse of the Hessian. Mackay [Mac92] uses numerical differentiation to calculate the Hessian of the error. An approximation of the Minimum Descripton Length principle that does not require calculation of the second derivatives is given by Weigend *et al.* [WHR90] ("Weight Elimination"), see also Barron and Barron, [BB88].

- **After learning: Network analysis.** One of the striking differences between connectionist modeling and traditional statistics is the larger number of parameters in neural networks. However, a key feature is that the potential number of parameters is often much larger than the effective number of parameters. Moody [Moo92] uses the Hessian of the error in a regularization framework to estimate the effective number of parameters of the network. Whereas Moody only considers the effective number of parameters at the end of the training process, when the error has reached a minimum, Weigend and Rumelhart [WR91] analyze the effective network size during training (via the dimension of the space spanned by the activations of the hidden units.) The gradual increase in network size with training time provides a justification for another heuristic for obtaining good generalization: to train a large (oversized) network, but stop training early.

Now, why a paper on second derivatives–isn't using second derivatives nothing but the chain rule for differentiation? Yes—but there are many ways, with varying degrees of efficiency and accuracy. To handle the complication of second derivatives over a complex network, Werbos, McAvoy and Su [WMS92] introduced the notion of the ordered derivative. Instead we use the notion of derivatives local to a unit and its immediate inputs/outputs, and derivatives global to the network, since these correspond to the key facets of the computation. Bishop [Bis92] avoided this complication by restricting results to sigmoidal units with error a simple sum and with no connections skipping layers.

The goal of this paper is to present exact and efficient methods for calculating second derivatives, and then to discuss a range of approximations: In Section 2, we define two classes of network structure, depending on whether the hidden units are like radial basis functions or like sigmoids. In Section 3, we derive the exact forms for each class and give parallel algorithms for the calculation. In Section 4, we systematically discuss various approximations and compare their accuracy and their computational complexity with the exact methods. Finally, in Section 5, we summarize the results and put them in context.

2 Notation

We only consider networks that are *directed acyclic graphs,* i.e., we do not consider recurrent networks that have feedback-loops. In a feedforward network, a connection from unit n to unit m means that unit m receives the activation from unit n during the forward pass. The activation value is often multiplied with a weight, $w_{m,n}$. The directed acyclic graph forms a partial order on the units in the network[3] that defines the order in which forward propagation should precede. If ordering of the units is consistent with this partial order, then it defines a suitable ordering for forward propagation through the units. Note such an ordering is not necessarily unique. If the reverse of an ordering is consistent with this partial order, then the ordering is suitable for backward propagation. In addition to the connectivity, specific activation functions for each unit have to be defined in order to specify the network.

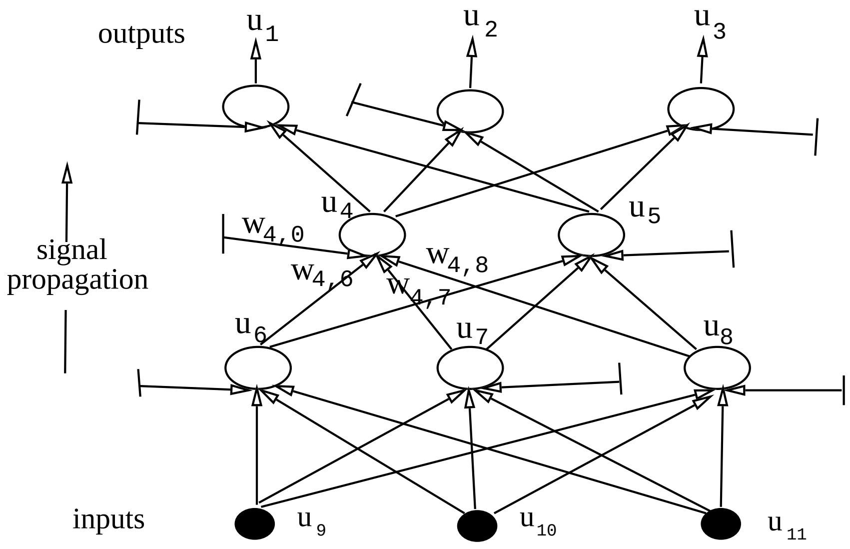

Figure 1: A simple network. Weights are shown as lines between units, biases as "terminated" lines.

In the simple example network shown in Figure 1, activations are passed upwards from unit to unit in the process of **forward propagation**. The activation of any unit $n \in \mathcal{N}$ is denoted by u_n which is a real number. The activation function for the unit m is a function of the activations u_n for units n feeding unit m, and the function is parameterized by a vector of parameters w_m. For instance, in Figure 1, u_4 is a function of the activations u_6, u_7 and u_8 and the weights $w_{4,6}$, $w_{4,7}$ and $w_{4,8}$ together with the bias $w_{4,0}$.

In the derivations below, it turns out to be advantageous to consider two classes of activation functions separately, allowing to derive exact algorithms. We define two types of *activation functions* for a unit n,

$$u_n = f_n(v_n, w_{n,0}) \quad \text{where} \quad v_n := \sum_{k \text{ connects to } n} \phi_{n,k}(u_k, w_{n,k}) \qquad \text{(r-type)} \qquad (1)$$

$$u_n = f_n(v_n) \quad \text{where} \quad v_n := w_{n,0} + \sum_{k \text{ connects to } n} u_k w_{n,k} \qquad \text{(s-type)} \qquad (2)$$

[3] *Partial order* means that for any pair of units i and j, either i comes before j, j comes before i, or we do not care about the order. Respectively, i is in an earlier layer, i is in a later layer or i and j are in identical layers.

for any functions $\phi_{n,k}$ and f_n.

An example of the r-type are radial basis function units which in general take a weighted sum of squared differences between their input and their center[4]. The radius could also be made dependent on the inputs k, to give $\sigma_{n,k}$. The s-type includes the usual sigmoid activation functions[5]. It is a dot-product with an offset (or bias or "0-weight") $w_{n,0}$ which can be thought of as the weight for a constant activation $u_0 = 1$.

Seemingly an exception to these two classes are so-called "Softmax" units (Bridle, [Bri89]). They are a convenient choice for the 1-of-N classification tasks: The constraints on probabilities to be non-negative and add up to one are modeled by using exponential units that are normalized, and this choice leads to a simple update rule in backpropagation. They can be expressed in the framework set up here by absorbing the normalization into the error function. Suppose there are N output units, $u_1, \ldots, u_N$, and the correct assignment for a pattern is to the i-th class, then its error E under cross-entropy is given by

$$E = -\log\left(\frac{e^{u_i}}{\sum_{1\leq j\leq N} e^{u_j}}\right) . \tag{3}$$

The error E for a single pattern depends on the outputs and targets for that pattern; typical error functions are mean squared error and cross-entropy. Details are given by Buntine and Weigend [BW91] and by Rumelhart *et al.* [RDGC93]. For many uses mentioned in the introduction we are interested in second derivatives for the sum (or average) error over the entire training set (the so-called "batch mode"), or at least a reasonable sized subset. Differentiation distributes over summation, so to calculate the derivatives of the total training set error we can calculate the derivatives of single patterns, and sum afterwards. We are thus primarily interested in

$$\frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}}$$

for the error E of a single pattern.

---

[4] A simple example is a spherically symmetric Gaussian radial basis function for unit n centered around $w_{n,k}$,

$$u_n \propto \exp\left(\frac{\sum_{k=1}^{d}\left(u_k - w_{n,k}\right)^2}{-2\sigma_n^2}\right) = \prod_{k=1}^{d}\exp\left(\frac{\left(u_k - w_{n,k}\right)^2}{-2\sigma_n^2}\right) .$$

The parameter σ_n describes the width (or standard-deviation) of the Gaussian and can be interpreted as the radius of the hyperspherical receptive field of the hidden unit in the d-dimensional input space. For accounting purposes, it can also be treated as a generalized "bias" term, $\sigma_n = w_{n,0}$. In this simple formula we assume spherical symmetry (σ_n is a scalar). This case is discussed further in Weigend *et al.* [WHR90].

[5] The sigmoid is a composition of two operators: an affine mapping giving v_n, followed by a nonlinear transformation f_n. First, the inputs into a hidden unit h are linearly combined, and an offset or bias $w_{n,0}$ is added. The sum is the *network-input* v_n,

$$v_n = \sum_k w_{n,0} + w_{n,k}x_k = w_{n,0} + \vec{w}_N \cdot \vec{u} \quad .$$

Interpreting this equation, the unit n only reponds to $\vec{w}_n \cdot \vec{u}$, i.e., the projection of the vector $\vec{u}$ onto the weight vector $\vec{w}_n = (w_{h1}, w_{h2}, ..., w_{hd})$. Changes in the input that are orthogonal to the direction of the weight vector have no effect on the activation of the hidden unit. The "equi-activation surfaces" (on which a hidden unit's activation is constant) are hyperplanes orthogonal to the direction of $\vec{w}_h$.
The second step can be viewed as "piping" v_n through a nonlinear *activation function* such as a sigmoid (or its symmetric version, a hyperbolic tangent) whose activation is given by

$$u_n = f_n(v_n) = \frac{1}{1+e^{-av_n}} = \frac{1}{2}\left(1+\tanh\frac{a}{2}v_n\right) \quad .$$

The sigmoid performs a smooth mapping $(-\infty, +\infty) \to (0, 1)$. The gain a can be absorbed into weights and biases without loss of generality and is set to unity.

The idea in the calculation of these second derivatives is the following: we need to arrive at global derivatives, but we want to obtain them by only using local derivatives (which are easily computed) and propagated local information. This is reflected in the notation: We treat the local derivatives of the activation functions (derivatives calculated from information local to the unit and its input activations) as "primitive components" in the calculation: rather than represent these as partial derivatives like $\partial u_n / \partial u_k$ or $\partial u_n / \partial w_{n,k}$, we write u_n^k and $u_n^{(k)}$, respectively. Similarly, the slope of the activation function, $\partial u_n / \partial v_n$ will be denoted by u_n'. Now v_n and $w_{n,k}$ are local information to the unit n so

$$\begin{aligned} u_n^{(k)} &:= \frac{\partial u_n}{\partial w_{n,k}} \\ u_n' &:= \frac{\partial u_n}{\partial v_n} \end{aligned}$$

However, in general

$$u_n^k \neq \frac{\partial u_n}{\partial u_k} \ ,$$

because u_k is non-local information. When some connections skip layers, the local derivative on the left hand-side and the global derivative on the right hand-side can differ. When k has only indirect connections to n, then the left hand-side will be zero but the right hand-side may be non-zero. So the distinction between the two forms must be maintained. Examples and the explicit formulae for r-type and s-type activations below will make this clearer. Werbos, McAvoy and Su [WMS92] use the notion of an ordered derivative to maintain the same distinction[6].

Notice that the superscript k represents partial differentiation w.r.t. the activation u_k, the superscript (k) represents partial differentiation w.r.t. the weight $w_{n,k}$, and the prime denotes partial differentiation w.r.t. the input sum v_n. The same spirit applies to shorthands such as $u_n^{k,l}$, $u_n^{(k),l}$ etc., and to the derivatives of the error function E. For instance, $E^{l,k}$ denotes the local partial derivative of the error E w.r.t. the output activations u_k and u_l which occur directly in E.

For instance, suppose the activation functions in Figure 1 are sigmoids. Then $u_4^4 = u_4^1 = 0$, $u_4' = u_4(1-u_4)$, $u_4'' = u_4(1-u_4)(1-2u_4)$, $u_4^6 = u_4(1-u_4)w_{4,6}$, $u_4^{(6)} = u_4(1-u_4)u_4$, $v_4^6 = w_{4,6}$, etc. Suppose there was also a direct connection from u_7 to u_2 skipping the middle layer. Then $u_2^7 = w_{2,7}$ but $\partial u_2/\partial u_7 = w_{2,7} + u_2^4 u_4^7 + u_2^5 u_5^7$. Locally, u_7 only effects u_2 through the weight $w_{2,7}$, but globally it also effects it indirectly through u_4 and u_5 as well. Hence the global and local derivatives can differ.

Also, by definition $u_n^k = u_n^{(k)} = u_n^{k,l} = u_n^{(k),l} = 0$ whenever unit k does not directly connect to unit n in the network, implying $u_n^n = 0$, etc. Likewise, E^m is zero if the activation u_m does not appear directly in E. For instance, suppose the energy function E is a direct function of the three network outputs in Figure 1. Then $E^7 = 0$, but $\partial E/\partial u_7$ might be non-zero because E^1 might be non-zero and u_1 is indirectly effected by u_7.

For the r-type activation functions, the local derivatives are:

$$\begin{aligned} u_n^k &= u_n' v_n^k \ , \\ u_n^{(k)} &= u_n' v_n^{(k)} \qquad \text{for } k \neq 0 \ , \\ u_n^{k,l} &= u_n'' v_n^k v_n^l + \delta_{k,l}\, u_n' v_n^{k,k} = \frac{u_n''}{u_n' u_n'} u_n^k u_n^l + \delta_{k,l}\, u_n' v_n^{k,k} \ , \\ u_n^{k,(l)} &= \frac{u_n''}{u_n' u_n'} u_n^k u_n^{(l)} + \delta_{k,l}\, u_n' v_n^{k,(k)} \qquad \text{for } l \neq 0 \ , \end{aligned}$$

[6] Their "+" derivative corresponds to our global derivative, and their non-superscripted derivative corresponds to our local derivative.

etc., where $\delta_{k,l}$ is the usual delta function (zero unless $k = l$). For s-type activation functions, the local derivatives are:

$$\begin{aligned} u_n^k &= u_n' w_{n,k} \ , \\ u_n^{(k)} &= u_n' u_k \ , \\ u_n^{k,l} &= u_n'' w_{n,k} w_{n,l} \ , \\ u_n^{(k),l} &= u_n'' u_k w_{n,l} \ . \end{aligned}$$

With the superscript notation for local derivatives of activation functions and error, we can express **backward propagation** of first derivatives with the following two Equations 4 and 5, and forward propagation with Equation 6. These equations apply to arbitrary networks in the form of directed acyclic graphs, including the case where connections skip layers,

$$\frac{\partial u_n}{\partial u_l} = \delta_{n,l} + \sum_k u_k^l \frac{\partial u_n}{\partial u_k} \qquad \text{(activation backward propagation)} \quad . \tag{4}$$

The factor u_k^l implies that the sum is only over those units k that l connects forward to. The term $\delta_{n,l}$ is usually not given in most formulations because it is implicity assumed that $n \neq l$. This equation is the chain rule of the backward pass in backpropagation. We propagate the partial derivatives of u_n backwards in the network starting at u_n and fanning backwards to the input units. For a particular n we repeatedly apply the equation to units l in the reverse order consistent with the network. A similar equation is used in error backpropagation[7]:

$$\frac{\partial E}{\partial u_m} = E^m + \sum_l \frac{\partial E}{\partial u_l} u_l^m \qquad \text{(error backward propagation)} \quad . \tag{5}$$

This expresses the change of the error as the activation of unit m is varied (while everything else is considered constant). The first term on the right hand side, E^m, represents the part where the error is directly affected by a change of u_m. This part only contributes if unit m is an output unit and is the usual base case in error backpropagation. The second term represents the usual recursive case of error backpropagation. This collects the partial derivatives that influence the error indirectly: it now uses the chain rule for the error, and the term u_l^m constrains the set of units l to those that m connects forward to.

The next equation is used in a forward propagation process to compute derivatives of the particular activation u_m. This is the simple chain rule of calculus.

$$\frac{\partial u_l}{\partial u_m} = \delta_{l,m} + \sum_k u_l^k \frac{\partial u_k}{\partial u_m} \qquad \text{(forward propagation)} \quad . \tag{6}$$

This equation is used to calculate partial derivatives of arbitrary activations w.r.t. the particular activation u_m. We propagate the derivatives w.r.t. u_m forwards in the network starting at units u_l that u_m directly connects forward to. For a particular m we repeatedly apply the equation to units l in an order consistent with the network.

The purpose of this section was to introduce the notation and to express the first-order backpropagation equations in this notation. We now turn to second-order derivatives.

3 Exact calculations

In this section, we present an exact algorithm for calculating second derivatives of the error for a single pattern. As mentioned in the previous section, we might later sum these over patterns

[7] The corresponding equation using the Werbos *et al.* notation of ordered derivatives is [WMS92, Equation (99)].

in a training set. We first give the general basic theorem. We then simplify it for the two cases of r-type and s-type units. The theorem is mainly included for logical completeness, and can be skipped by most readers. We close this section with a procedural interpretation.

When reading the theorem and the two corollaries, bear in mind that the partial derivatives represented with the "∂" sign will have to be calculated using forward or backward propagation passes through the network, whereas implicit partial derivatives in the form E^l or $u_l^{m,(k)}$, as described previously, are available directly from information local to each unit. Also, please note that the summations use the convention described at the end of the last section, for instance, described for Equation (4). Due to $E^{m,l}$ the first term in Equation (7) sums over output units l appearing directly in E. Likewise, due to u_l^m the second term sums over activation units l that m connects to.

Basic theory

If we assume nothing about the form of the activation functions, we can apply the chain rule to obtain a general form of the second derivative. While we do not advocate applying this directly, we use it to derive exact algorithms for the r-type and s-type cases introduced in Equation 1.

Theorem 1 *Assume a neural network framework as set up previously with arbitrary acyclic connections and activation functions, and with E a function of activations. Equations (7) and (8) assume there is no sequence of connections from m to n (i.e. u_n is not a partial function of u_m) although m and n may be equal*[8].

$$\frac{\partial^2 E}{\partial u_m \partial u_n} = \sum_l E^{m,l} \frac{\partial u_l}{\partial u_n} + \sum_l \frac{\partial^2 E}{\partial u_l \partial u_n} u_l^m + \sum_{k,l} \frac{\partial E}{\partial u_l} u_l^{k,m} \frac{\partial u_k}{\partial u_n} , \tag{7}$$

$$\frac{\partial^2 E}{\partial w_{m,i} \partial w_{n,j}} = \frac{\partial^2 E}{\partial u_m \partial u_n} u_m^{(i)} u_n^{(j)} + \delta_{m,n} \frac{\partial E}{\partial u_m} u_m^{(i),(j)} + \frac{\partial E}{\partial u_m} \left(\sum_l u_m^{l,(i)} \frac{\partial u_l}{\partial u_n} \right) u_n^{(j)} , \tag{8}$$

where the delta function $\delta_{m,n}$ takes the value 1 if $m = n$ and 0 otherwise.

Proof Sketch Equation (7) follows by differentiating the error backward propagation Equation (5) with respect to u_n. Notice E^m will be zero if the activation u_m does not appear directly in E. Also,

$$\frac{\partial E^m}{\partial u_n} = \sum_l E^{m,l} \frac{\partial u_l}{\partial u_n}$$

Equation (8) follows by differentiating

$$\frac{\partial E}{\partial w_{m,i}} = \frac{\partial E}{\partial u_m} u_m^{(i)}$$

with respect to $w_{n,j}$. Notice the final summation in Equation (8) is only present if there is a sequence of connections from unit n to unit m. □

Equation (7) represents a recursive formula for calculating second derivatives of the error function with respect to the activation functions. For fixed n, the recursion starts with m at the output of the network and works its way down while m has no connections to n. These second derivatives are a function of first derivatives in the network and the second derivatives from connected units. In general these can be calculated in backward and forward propagation passes, which we describe below. Equation (8) uses the results of Equation (7) to calculate second derivatives with respect to weights.

[8] This assumption is not a constraint. Simply swap n and m if the theorem does not apply directly.

Assumptions on the form of the activation (r-type or s-type) can simplify these equations. The main simplification comes about because for both r-type and s-type units, $u_l^{k,m} = 0$ whenever $k \neq m$. This reduces computation by a factor equal to the network fan-in (maximum number of units connecting forward into a unit). In the corollary, we use the same summation conventions as just described, so the first sum in Equation (12) is over units l that m connects forward to, and the second sum is over output units l. Similarly, the first sum in Equation (10) is over output units l and k, and the second sum is over any unit l that both m and n connect forward to directly or indirectly (typically, later in the network).

Corollary 1 (r-type) *Assume that for every unit activation input is r-type. Consider how the above second derivatives evaluate. Equation (9) assumes there is no sequence of connections from m to n. (So $\partial u_n / \partial u_m = 0$ but $\partial u_n / \partial u_m$ is possibly non-zero.)*

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{m,i} \partial w_{n,j}} \quad = \quad & \frac{\partial^2 E}{\partial v_m \partial v_n} v_m^{(i)} v_n^{(j)} \ + \ 1_{m \neq n} 1_{i=0} \frac{\partial E}{\partial u_m} \left(\frac{u'^{(0)}_m}{u'_m u_m^{(0)}} - \frac{u''_m}{u'_m u'_m} \right) \frac{\partial u_m}{\partial u_n} u_m^{(0)} u_n^{(j)} \\ & + \ 1_{m \neq n} 1_{i \neq 0} \frac{\partial E}{\partial u_m} u'_m v_m^{i,(i)} \frac{\partial u_i}{\partial u_n} u_n^{(j)} \ + \ 1_{m=n} \frac{\partial E}{\partial u_m} \left(u_m^{(i),(j)} - \frac{u''_m}{u'_m u'_m} u_m^{(i)} u_m^{(j)} \right) ,\end{aligned} \tag{9}$$

where the indicator function $1_{m \neq n}$ takes the value 1 if $m \neq n$ and 0 otherwise, etc., and

$$\frac{\partial^2 E}{\partial v_m \partial v_n} \quad = \quad u'_m u'_n \left(\sum_{l,k} E^{l,k} \frac{\partial u_l}{\partial u_m} \frac{\partial u_k}{\partial u_n} \ + \ \sum_l G_l \frac{\partial u_l}{\partial u_m} \frac{\partial u_l}{\partial u_n} \right) , \tag{10}$$

$$G_m \quad \equiv \quad \frac{\partial E}{\partial u_m} \frac{u''_m}{u'_m u'_m} \ + \ \sum_l \frac{\partial E}{\partial u_l} u'_l v_l^{m,m} . \tag{11}$$

Equation (10) can also be represented in recursive form.

$$\frac{\partial^2 E}{\partial v_m \partial v_n} \ = \ G_m u'_m u'_n \frac{\partial u_m}{\partial u_n} \ + \ u'_m \sum_l \frac{\partial^2 E}{\partial v_n \partial v_l} v_l^m \ + \ u'_m u'_n \sum_l E^{m,l} \frac{\partial u_l}{\partial u_n} . \tag{12}$$

When unit m connects forward to no other units (i.e. is not an output), the second term disappears, so we have the base case. When unit m does not appear directly in the error E (i.e. is not an output), then the third term disappears. When the error E is mean square error, $E^{m,l} = 0$ for $m \neq l$, the third term reduces to: $u'_m u'_n \partial u_m / \partial u_n$. When both units m and n connect forward to no other units (i.e. both are output), then

$$\frac{\partial^2 E}{\partial v_m \partial v_n} \ = \ \delta_{m,n} E^m u''_m \ + \ E^{m,n} u'_m u'_n . \tag{13}$$

Proof Sketch When there are no sequence of connections from unit m to n,

$$\begin{aligned}\frac{\partial E}{\partial v_m} \quad &= \quad u'_m \frac{\partial E}{\partial u_m} , \\ \frac{\partial^2 E}{\partial v_m \partial v_n} \quad &= \quad u'_m u'_n \frac{\partial^2 E}{\partial u_m \partial u_n} + \frac{\partial E}{\partial u_m} \frac{u''_m u'_n}{u'_m} \frac{\partial u_m}{\partial u_n} .\end{aligned} \tag{14}$$

Rearranging and substituting both second derivatives in Equation (7) yields Equation (12) and the definition of G_m. To simply the definition of G_m requires the following:

$$\begin{aligned}u_l^{k,m} \quad &= \quad u''_l v_l^k v_l^m + \delta_{k,m} u'_l v_l^{m,m} , \\ \sum_k u_l^{k,m} \frac{\partial u_k}{\partial u_n} \quad &= \quad u'_l v_l^{m,m} \frac{\partial u_m}{\partial u_n} + \frac{u''_l v_l^m}{u'_l} \frac{\partial u_l}{\partial u_n} .\end{aligned}$$

Second, we can prove by induction from Equation (7) and the definition of G_m that

$$\frac{\partial^2 E}{\partial u_m \partial u_n} = \sum_{l,k} E^{l,k} \frac{\partial u_l}{\partial u_m} \frac{\partial u_k}{\partial u_n} + \sum_l G_l \frac{\partial u_l}{\partial u_m} \frac{\partial u_l}{\partial u_n} - \frac{\partial E}{\partial u_m} \frac{u''_m}{u'_m u'_m} \frac{\partial u_m}{\partial u_n} .$$

Manipulation of this and Equation (14) yields Equation (10).

Substituting in Equation (14) into Equation (9), and cancelling terms almost gives us Equation (8). Equivalence of Equations (9) and (8) is shown by first simplifying

$$\begin{aligned} \sum_l u_m^{l,(i)} \frac{\partial u_l}{\partial u_n} &= 1_{m \neq n} \frac{u''_m}{u'_m u'_m} \frac{\partial u_m}{\partial u_n} u_m^{(i)} + u'_m v_m^{i,(i)} \frac{\partial u_i}{\partial u_n} && \text{for } i \neq 0 \\ &= 1_{m \neq n} \frac{u'^{(0)}_m}{u'_m} \frac{\partial u_m}{\partial u_n} && \text{for } i = 0 \end{aligned}$$

□

The middle two terms in Equation (9) correspond to the cases where unit n connects directly or indirectly to unit m. The last term corresponds to the case where units n and m are the same.

Initialization of the recursion for Equation (12) (the "base case") occurs when unit m is an output unit which does not connect further to other units. For this the second term in Equation (12) vanishes, so no recursion exists. Also the third term $E^{m,l} = 0$ if m is not an output unit. For most error functions the third term is easy to compute since for many error functions $E^{m,l} = 0$ for $m \neq l$. For cross entropy error, or when using the Softmax units at the output as in Equation (3), the third term requires summation over output units l. Second derivatives are therefore only expensive to calculate exactly if both weights are at hidden units.

If one unit is in the final hidden layer (immediately before the output layer), then Equation (10) can be used directly instead of Equation (12). In this case the sums in Equation (10) are over the output units so can be computed directly. This sum may be faster than the recursion if the calculation is done in parallel with that of the output units.

Notice that if units use activation functions with s-type such as a sigmoid, then the corollary simplifies further because $u'^{(0)}_m = u''_m$, $v_l^{m,m} = 0$, $v_l^{m,(m)} = 1$ if m connects directly to l, etc. Equation (12) remains unchanged but the other two simplify. Werbos *et al.* present a formulation using ordered derivatives, and Bishop [Bis92] presents the simplified case where there are no connections skipping layers, and the error E is a sum over network outputs (i.e. $E^{m,l} = 0$ for $m \neq l$). Equation (12) corresponds to Bishop's Equation (2.17) and Equation (10) corresponds to Bishop's Equation (3.2).

Corollary 2 (s-type) *Assume that for every unit activation input is s-type and there is no sequence of connections from units m to n. Then the formulae in Corollary 1 evaluate to:*

$$\begin{aligned} G_m &= \frac{\partial E}{\partial u_m} \frac{u''_m}{u'_m u'_m} , \\ \frac{\partial^2 E}{\partial w_{m,i} \partial w_{n,j}} &= \frac{\partial^2 E}{\partial v_m \partial v_n} v_m^{(i)} v_n^{(j)} + 1_{i \neq 0} \frac{\partial E}{\partial u_m} u'_m \frac{\partial u_i}{\partial u_n} u_n^{(j)} , \end{aligned} \tag{15}$$

where $\partial^2 E / \partial v_m \partial v_n$ is calculated as before using Equation (12) or (10).

Algorithms

The algorithm below applies to both the r-type and s-type activation functions. Each individual step can be parallelized at the unit level. The algorithm for the general case is similar, except we drop Step 2 and the corresponding equations from the theorem are used instead of the simplifications.

1. Run standard backpropagation to compute the partial derivatives $\partial E / \partial u_n$.

2. Thus compute G_m and store this local to each unit. This will be constant time for each s-type unit, and require time proportional to the fan-out (number of units that the unit connects to) for each r-type unit.

3. For each unit n calculate $\partial^2 E / \partial w_{m,i} \partial w_{n,j}$ for units m after n in some order consistent with the network. This will require storage local to each unit m of three values.

   (a) Calculate derivatives $\partial u_m / \partial u_n$ for each unit m after n in the chosen order consistent with the network. Start at units that receive activation from unit n and propagate forward using Equation (6). For units m that do not have a connection from n directly or indirectly, $\partial u_m / \partial u_n = 0$. If unit n connects forward to no other units (i.e. is an output) then this step need not be done. Otherwise, this step takes approximately one forward propagation cycle.

   (b) Backward propagate starting from the output units to calculate $\partial^2 E / \partial w_{m,i} \partial w_{n,j}$ for each unit m after n using Equation (12). (Alternatively, if there is at most 1 hidden layer, calculate using Equation (10) for all units in parallel.) The discussion in Corollary 1 describes various simplifications. This step takes approximately one backpropagation cycle if n is not an output unit.

   (c) The second derivatives w.r.t. the weights can now be calculated directly using Equations (9) or (15), and the store for $\partial^2 E / \partial v_m \partial v_n$ and $\partial u_m / \partial u_n$ deleted. This step can be done in parallel with the previous. The computation for each node m goes with the square of the fan-in.

For instance, consider the simple network in Figure 1. The propagations done in Step 3a for the unit $n = 8$ are displayed in Figure 2. The propagations done in Step 3b for the unit $n = 8$

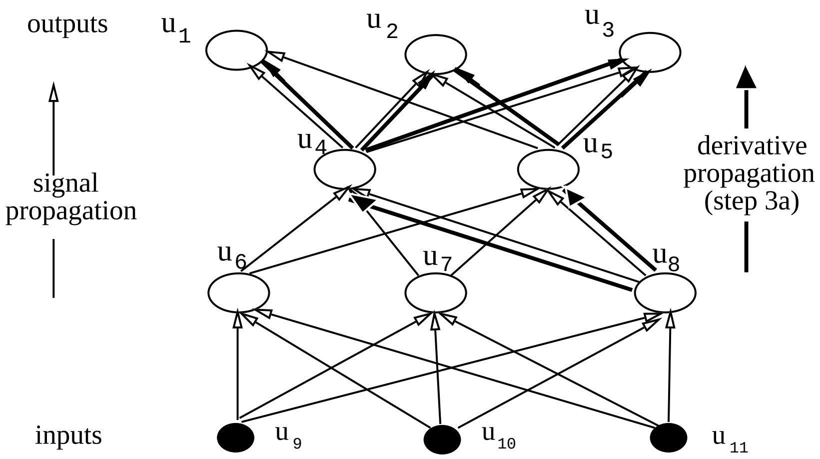

Figure 2: Forward propagation to calculate derivatives w.r.t. u_8

are displayed in Figure 3.

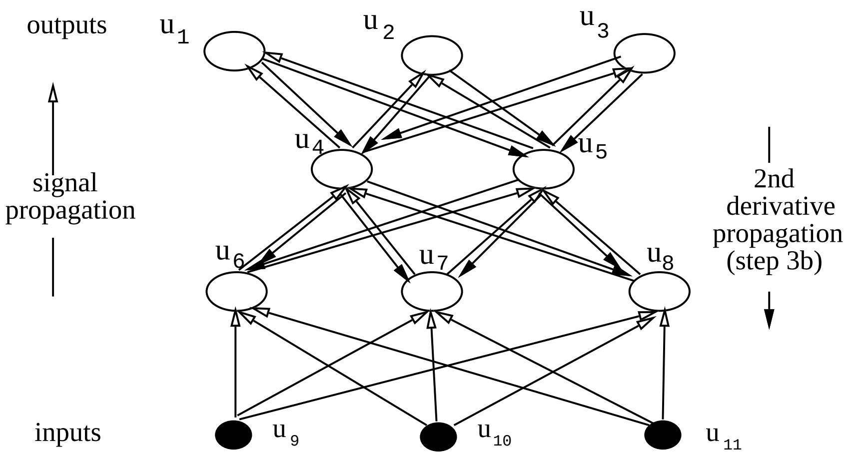

Figure 3: Backward propagation to calculate derivatives w.r.t. u_8

To derive the necessary components, this calculation requires approximately $2h+2$ back/forward-propagation passes, where h is the number of hidden units in the network. The final calculation of second derivatives w.r.t. the weights is unavoidably of the order of $|w|^2$, for w the number of weights in the network, since that is the size of the Hessian. This is the dominant term in the calculation.

Notice that for the special case where we only use the block diagonals, for instance, as assumed by El-Jaroudi and Makhoul [EJM90], $\partial^2 E/\partial w_{m,i}/\partial w_{n,j}$ for $n = m$, of a network, Equation (10) should be used. Thus Equation (10) replaces Equation (12) in Step 3(b) and the calculation is only done for the unit $m = n$. Likewise for Step 3(c).

4 Approximations

4.1 Ignoring terms

If we have a layered network with no connections spanning multiple layers, and we assume units m and n are at the same level, then from Equation (7) we get

$$\frac{\partial^2 E}{\partial u_m \partial u_n} = E^{m,n} + \sum_{l,k} \frac{\partial^2 E}{\partial u_l \partial u_k} u_l^m u_k^n + \sum_l \frac{\partial E}{\partial u_l} u_l^{m,n} .$$

Le Cun *et al.*'s approximation corresponds to the case where we have s-type units and we assume these second derivatives are zero for units $m \neq n$ (so $R_{m,n} = 0$), giving the rule (they use a different parameterization so their form is different):

$$\begin{aligned} \frac{\partial^2 E}{\partial u_m^2} &\approx \sum_l \left(\frac{\partial^2 E}{\partial u_l^2} u_l^m u_l^m + \frac{\partial E}{\partial u_l} u_l'' w_{l,m}^2 \right) , \\ \frac{\partial^2 E}{\partial w_{m,i}^2} &\approx \left(\frac{\partial^2 E}{\partial u_m^2} u_m' u_m' + \frac{\partial E}{\partial u_m} u_m'' \right) u_i^2 . \end{aligned}$$

These can be computed with one backpropagation pass, so calculation is efficient [LDS90]. MacKay, however, found the method inaccurate for his purposes [Mac92] and instead dropped the first term from Equation (12) and unfolds the recursion leaving a calculation that requires only the first derivatives,

$$\frac{\partial^2 E}{\partial w_{m,i} \partial w_{n,j}} \approx \sum_{l,k} E^{l,k} \frac{\partial u_l}{\partial u_m} \frac{\partial u_k}{\partial u_n} u_m^{(i)} u_n^{(j)} .$$

He reports this is inaccurate when approximating the determinant of the matrix of second derivatives, or looking at individual second derivatives, but seems fine when approximating the trace of the matrix. This approximation, ignoring the second derivatives, corresponds to the Levenberg-Marquardt approximation in non-linear least squares [PFTV86, Section 14.4]. Clearly, better approximations terms are available that ignore a few less terms.

4.2 Scoring

A second form of approximation exists if the network error function being minimized corresponds to the negative logarithm of the likelihood of the training sample, as is often the case when using mean-square error or cross-entropy error functions [BW91, BW87, EJM90]. Suppose the likelihood of the training sample of size D is given as a product over patterns (x_d, y_d) in the sample

$$p(\vec{y} \mid \vec{x}, w) = \prod_{d=1,\ldots,D} l(y_d \mid x_d, w) ,$$

where $l(y_i \mid x_i, w)$ is the likelihood of the d-th pattern and is a function of the network output for input x_i and the "correct" output y_i. Then maximum likelihood training (maximum *a posterior* training is similar) corresponds to minimizing the error function with component for each pattern

$$E_d = -\log l(y_d \mid x_d, w) .$$

For instance, when using the mean-square error error function on a single output, we are implicitly using the Gaussian likelihood

$$l(y_d \mid x_d, w) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_d - o_d)^2\right) ,$$

where σ is the standard deviation and o_d is the network output for input x_d.

When network weights are near a local maxima of the likelihood the second derivatives summed over the full set of training patterns can be approximated using the following formulae:

$$\begin{aligned} \sum_{d=1,\ldots,D} \frac{\partial^2 E_d}{\partial w_{m,i} \partial w_{n,j}} &= \frac{\partial^2 - \log p(\vec{y} \mid \vec{x}, w)}{\partial w_{m,i} \partial w_{n,j}} \\ &= \sum_{d=1,\ldots,D} \frac{\partial^2 - \log l(y_d \mid x_d, w)}{\partial w_{m,i} \partial w_{n,j}} \\ &\approx - \sum_{d=1,\ldots,D} \frac{\partial - \log l(y_d \mid x_d, w)}{\partial w_{m,i}} \frac{\partial - \log l(y_d \mid x_d, w)}{\partial w_{n,j}} \\ &= - \sum_{d=1,\ldots,D} \frac{\partial E_d}{\partial w_{m,i}} \frac{\partial E_d}{\partial w_{n,j}} . \end{aligned} \tag{16}$$

Notice this only requires calculation of the first derivatives of the error. The approximation step from line 2 to 3 is justified because at a local maxima the two sides are approximately equal on the assumption that the weights w are approximately correct (which they may well not be) and the sample size D is large. This follows from the probabilistic identity

$$\int \frac{\partial \log p(z|\theta)}{\partial \theta_1} \frac{\partial \log p(z|\theta)}{\partial \theta_2} p(z|\theta) \, \mathrm{d}z = - \int \frac{\partial^2 \log p(z|\theta)}{\partial \theta_1 \partial \theta_2} p(z|\theta) \, \mathrm{d}z ,$$

for the probability density function $p(z|\theta)$ parameterized by a vector of parameters θ. This approximation is used in "Fisher's scoring method" for maximum likelihood training [MN89]. It is best used during search when fast estimates of second derivatives are required (see, for instance, [PFTV86, Section 14.4]). The approximation would be misleading when good approximations for error bars are required because it assumes that the thing we are attempting to evaluate is approximately true (that the weights are correct).

With this approximation, a sequential calculation for the inverse of the second derivative is also available, as given by Hassibi and Stork ([HS93]). The Sherman-Morrison-Woodbury formula for iterative matrix inversion [GV89, PFTV86] is:

$$(A + b \cdot d^T)^{-1} \;=\; A^{-1} - \frac{A^{-1} \cdot b \cdot d^T \cdot A^{-1}}{(1 + d^T \cdot A^{-1} \cdot b)} \;,$$

where A is a square matrix and b and d are column vectors of the same size. This is used to sequentially calculate the inverse of

$$H_p \;=\; \sum_{d=1,\ldots,p} \frac{\mathrm{d}E_d}{\mathrm{d}w} \frac{\mathrm{d}E_d}{\mathrm{d}w}^T \;,$$

where $\mathrm{d}E_d/\mathrm{d}w$ is the column vector of partial derivatives $\partial E_d/\partial w_{m,i}$. With the formula

$$H_{p+1}^{-1} \;=\; H_p^{-1} - \frac{H_p^{-1} \cdot \frac{\mathrm{d}E_{p+1}}{\mathrm{d}w} \cdot \frac{\mathrm{d}E_{p+1}}{\mathrm{d}w}^T \cdot H_p^{-1}}{\left(1 + \frac{\mathrm{d}E_{p+1}}{\mathrm{d}w}^T \cdot H_p^{-1} \cdot \frac{\mathrm{d}E_{p+1}}{\mathrm{d}w}\right)} \;,$$

H_p never needs to be computed. To initialize the calculation H_0 is set to some tiny matrix whose inverse is known. Then H_{p+1}^{-1} is computed from H_p^{-1}, etc., eventually yielding

$$\left(\sum_{d=1,\ldots,D} \frac{\mathrm{d}^2 E_d}{\mathrm{d}w \mathrm{d}w} \right)^{-1} \;=\; H_D^{-1} \;,$$

in order $D|w|^2$ operations where $|w|$ is the number of weights. Clearly, if the number of weights is much larger than the number of patterns, then it would be more efficient to calculate the inverse once at the end using standard matrix inversion methods. However, networks are sometimes trained with many more weights than input patterns.

4.3 Numerical differentiation

A third more exact approximation of second derivatives can be done by numerical differentiation of the first derivatives, which in turn can be calculated using standard backpropagation with Equation (5). MacKay has done this by numerical differentiation of the derivatives w.r.t. the weights [Mac92]. If there are $|w|$ different weights in the network, then this requires $|w| + 1$ backpropagation passes to compute the necessary first derivatives (although many intermediate results can be cached so full passes are not needed). A more efficient approach is to use numerical differentiation to calculate second derivatives of the activations in $h + 1$ backpropagation passes, where h is the number of hidden units. (These partial derivatives are readily available if n or m is an output unit, see comments after Corollary 1, and are not needed for input units.) Assuming there are no sequence of connections from unit m to unit n, this uses the approximation

$$\frac{\partial^2 E}{\partial u_m \partial u_n} \;\approx\; \frac{1}{\Delta u_n} \left(\left. \frac{\partial E}{\partial u_m} \right|_{u_n = u_n + \Delta u_n} - \frac{\partial E}{\partial u_m} \right) \;,$$

for some small value $\triangle u_n$. The first derivative on the right hand side is made by forcing $u_n = u_n + \triangle u_n$ in the calculation. The corresponding formula can be applied to compute second derivatives w.r.t. v_m and v_n instead of u_m and u_n. The second derivatives can then be found using Equation (8) in the general case, or Equations (9) and (15). Of course, additional first derivatives $\partial u_m / \partial u_n$ and $\partial u_i / \partial u_n$ are required and can be got in approximately h backpropagation passes. A similar method is suggested by Becker and Le Cun [BL88]. This approximation is therefore of the same computational order as the exact calculations described previously, but requires little additional algorithm overhead other than the backpropagation algorithm.

Moller and Pearlmutter have applied this numerical approach to calculate the second derivatives along a particular arc [Mo93b, Pea93] in 2 backpropagation passes. Let v be a column vector in weight space expressing a direction of interest, and let $\triangle$ be some small value.

$$\frac{\mathrm{d}^2 E}{\mathrm{d}w \mathrm{d}w} \cdot v \approx \frac{1}{\triangle} \left(\left. \frac{\mathrm{d}E}{\mathrm{d}w} \right|_{w = w + v\triangle} - \frac{\mathrm{d}E}{\mathrm{d}w} \right) ,$$

5 Summary

In general, the calculation of the full Hessian of a network of $|w|$ weights is an $O(|w|^2)$ operation since the matrix itself has $O(|w|^2)$ entries. If the Hessian is required for the batch error, then this calculation needs to be repeated for each pattern. Exact calculation of the necessary intermediate terms for r-type or s-type units in arbitrarily connected networks takes approximately $2h + 2$ backward/forward propagation passes. Numerical differentiation offers a simpler implementation and requires $2h + 1$ backward propagation passes if differentiation is done for activations, or $|w| + 1$ backward propagation passes if differentiation is done for weights. This, however, may require experimentation in setting the right $\triangle$. The scoring approximation is the simplest because all that is required is one backward propagation pass. However, it is really only justified when the network is at the global minimum of the batch error.

A block diagonal approximation can be made to the Hessian, where weights at different nodes are assumed to have no interaction in the Hessian. The Hessian then consists of a number of blocks, one for each node. Simplifications of each of the above methods could easily be developed.

Finally, the product of the Hessian by a vector, as required in some indirect second-order training methods, can be calculated either exactly or by numerical differentiation.

Acknowledgements

Thanks to Sue Becker, Yann Le Cun, David MacKay and Martin Møller for useful feedback.

References

[BB88] A.R. Barron and R.L. Barron. Statistical learning networks: A unifying view. In *INTERFACE'88 – 20th Symposium on the Interface: Computing Science and Statistics*, pages 192–203. American Statistical Association, 1988.

[Bis92] C. Bishop. Exact calculation of the Hessian matrix for the multilayer perceptron. *Neural Computation*, 4:494–501, 1992.

[BL88] S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second order methods. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37. Morgan Kaufmann, 1988.

[Bri89] J.S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman-Soulié and J. Hérault, editors, *Neuro-computing: Algorithms, Architectures and Applications*, pages 223–236. Springer-Verlag, 1989.

[BW87] E.B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D.Z. Anderson, editor, *Neural Information Processing Systems (NIPS)*, pages 52–61, 1987.

[BW91] W.L. Buntine and A.S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[EJM90] A. El-Jaroudi and J. Makhoul. A new error criterion for posterior probability estimation with neural nets. In *Int. Joint Conf. on Neural Networks*, pages III–185–192, San Diego, CA, 1990.

[Fah88] S.E. Fahlman. Faster-learning variations on backpropagation: an empirical study. In D.S. Touretzky, G.E. Hinton, and T.J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, 1988.

[GV89] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, second edition, 1989.

[HS93] B. Hassibi and D.G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S.J. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems 5 (NIPS*92)*. Morgan Kaufmann, 1993.

[Jac88] R.A. Jacobs. Increased rates of convergence through learning rate adaption. *Neural Networks*, 1:295–307, 1988.

[LDS90] Y. Le Cun, J.S. Denker, and S.A. Solla. Optimal brain damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS*89)*, pages 598–605. Morgan Kaufmann, 1990.

[LKS91] Y. Le Cun, I. Kanter, and S.A. Solla. Second order properties of error surfaces: Learning time and generalization. In R.P. Lippmann, J. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*, pages 918–924. Morgan Kaufmann, 1991.

[LSP93] Y. Le Cun, P.Y. Simard, and B.A. Pearlmutter. Automatic learning rate maximization in large adaptive machines. In S.J. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems 5 (NIPS*92)*. Morgan Kaufmann, 1993.

[Mac92] D.J.C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448-472, 1992.

[Mo93a] M.F. Møller. *Exact Calculation of the Product of the Hessian Matrix of Feed-Forward Network Error Functions and a Vector in O(N) Time.* Technical Report PB-432, Computer Science Department, Aarhus University, Denmark, 1993.

[Mo93b] M.F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 1993. To appear.

[MN89] P. McCullagh and J.A. Nelder. *Generalized Linear Models.* Chapman and Hall, London, second edition, 1989.

[Moo92] J.E. Moody. The *effective* number of parameters: an analysis of generalization and regularization in nonlinear learning systems. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4 (NIPS*91)*, pages 847–854. Morgan Kaufmann, 1992.

[Pea93] B.A. Pearlmutter. Fast exact multiplication by the Hessian. *Submitted to Neural Computation*, 1993.

[PFTV86] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 1986.

[RDGC93] D.E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. Backpropagation: The basic theory. 1993. In Y. Chauvin and D.E. Rumelhart, *Backpropagation: Theory, Architectures and Applications.* Lawrence Erlbaum, 1993.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, pages 318–362. MIT Press, 1986.

[WHR90] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart. Predicting the future: a connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[WMS92] P.J. Werbos, T. McAvoy, and T. Su. Neural networks, system identification, and control in the chemical process industry. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control*, pages 283–356. Van Nostrand Reinhold, 1992.

[WR91] A.S. Weigend and D.E. Rumelhart. Generalization through minimal networks with application to forecasting. In E.M. Keramidas, editor, *INTERFACE'91 – 23rd Symposium on the Interface: Computing Science and Statistics*, pages 362–370. Interface Foundation of North America, 1991.