NISTIR 8037

NEURBT: A Program for Computing Neural Networks for Classification using Batch Learning

Javier Bernal

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8037

National Institute of
Standards and Technology
U.S. Department of Commerce

NISTIR 8037

NEURBT: A Program for Computing Neural Networks for Classification using Batch Learning

Javier Bernal
Applied and Computational Mathematics Division
Information Technology Laboratory

This publication is available free of charge from:
http://dx.doi.org/10.6028/NIST.IR.8037

February 2015

U.S. Department of Commerce
Penny Pritzker, Secretary

National Institute of Standards and Technology
Willie May, Acting Under Secretary of Commerce for Standards and Technology and Acting Director

NEURBT: A Program for Computing Neural Networks for Classification using Batch Learning

Javier Bernal

National Institute of Standards and Technology,

Gaithersburg, MD 20899, USA

Abstract

NEURBT, a Fortran 77 program for computing neural networks for classification using batch learning, is discussed. NEURBT is based on Møller's scaled conjugate gradient algorithm which is a variation of the traditional conjugate gradient method, better suited for the non-quadratic nature of neural networks. Different aspects of the implementation are discussed such as the efficient computation of gradients and multiplication of vectors by Hessian matrices that are required by Møller's algorithm, and the stochastic (re)initialization of weights.

1 Introduction

Neural networks are computational models that work by simulating the way the brain processes information. They are often used to recognize patterns in a data set. Once the neural network is trained on sample patterns of the data, it can then be used for attempting to recognize other patterns as they are fed through the network.

Let A be a set of points or pattern vectors in Euclidean $d-$dimensional space that is partitioned into n classes. The basic structure of a neural network consists of layers or columns of mostly computing nodes, or neurons, arranged from left to right (see Figure 1) in such a way that the result of a

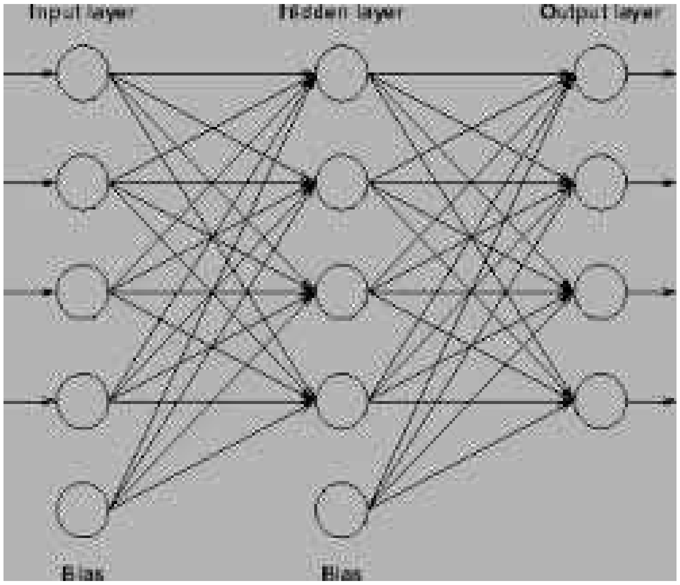

Figure 1: A 3−layer neural network. The leftmost layer is the input layer. From left to right in the network, the middle layer is the first layer with computing neurons and as such is called the first layer of the network. It is a hidden layer as well. The rightmost layer is the output layer. The input layer and the first layer have bias neurons (bottom neurons).

computation at each neuron in a layer contributes to the input of neurons in the next layer. The layer at the extreme left of the network is called the input layer of the network (see Figure 1) and consists of $d+1$ neurons. A pattern vector in A, say $a = \{a_k\}$, $k = 1, \dots, d$, is introduced into the network through the input layer as follows: a is augmented to be of dimension $d+1$ by setting a_{d+1} equal to 1; neurons in the input layer are labeled with integers from 1 to $d+1$; and for each k, $k = 1, \dots, d+1$, coordinate a_k is assigned to neuron k (neuron with label k) and as such interpreted to be the output of neuron k (neuron $d+1$ is called a bias neuron and its output is 1 for all patterns). The layer immediately to the right of the input layer, unlike the input layer, consists of computing neurons (except for the last neuron which is a bias neuron), and is called the first layer of the network. Like the input layer, it has $d+1$ neurons which are then labeled with integers from $d+2$ to $2d+2$. Given integer i, $d+2 \leq i \leq 2d+1$, a number x_i is designated the input to neuron i (in the first layer) which is a weighted sum of the outputs of the input layer (the coordinates of the augmented pattern a) expressed as $x_i = \sum_{k=1}^{d+1} w_{ki} a_k$. Here for each k, $k = 1, \ldots, d+1$, w_{ki} is the weight modifying the pattern coordinate a_k before it is fed into neuron i (as part of x_i). In order to make neuron i into a computing neuron, the sigmoid activation function $\sigma(x) = 1/(1+e^{-x})$ is assigned to it. σ is a function with derivatives of all orders and values between 0 and 1. $y_i = \sigma(x_i)$ is then designated the ouput of neuron i, $d+2 \leq i \leq 2d+1$, while $y_{2d+2} = 1$ is designated the output of neuron $2d+2$ (the bias neuron). Inductively, given layers M and L, consecutive layers in the network from left to right; $\{y_m\}$, the set of outputs of neurons in layer M; l_1, l_2, $l_1 < l_2$, integers such that neurons in layer L are labeled with integers from l_1 to l_2; and neuron l, a neuron in layer L, $l_1 \leq l \leq l_2 - 1$; then a number x_l is designated the input to neuron l which is a weighted sum of the outputs of layer M expressed as $x_l = \sum_m w_{ml} y_m$. In addition the same sigmoid activation function σ defined above is assigned to neuron l and $y_l = \sigma(x_l)$ is designated the output of neuron l, $l_1 \leq l \leq l_2 - 1$, while $y_{l_2} = 1$ is designated the ouput of neuron l_2 (the bias neuron of layer L).

With the exclusion of the layer at the extreme right of the network, layers to the right of the input layer are called hidden layers (in Figure 1 the middle layer, i.e., the first layer, is the only hidden layer), and hidden layers to the right of the first layer (there are none in the network of Figure 1) are assumed to consist of the same number of neurons, a number greater than 1 and preferably greater than d and n. The layer at the extreme right of the network is called the output layer of the network (see Figure 1). For consistency with definitions above involving consecutive layers L and M we assume at first that the output layer contains a bias neuron besides n computing neurons. As it will become apparent below there is a one-to-one correspondence between the n computing neurons in this layer and the classes into which the set A of patterns is partitioned. Reducing the number of neurons in the output layer to n by dropping the dummy bias neuron in the layer and letting nq be the total number of neurons in the network, neurons in the output layer are then labeled with integers from $nq - n + 1$ to nq. Additionally, letting nw be the total number of weights in the network, a natural order can be established for weights so that any given set of nw weights can be uniquely identified with a vector, a weight vector, in weight space, the Euclidean space of dimension nw, and vice versa.

Given a pattern a in A, then for some q, $1 \leq q \leq n$, a is in class q, and an $n-$dimensional vector $r(a) = \{r(a)_m\}$, $m = 1, \ldots, n$, called the desired response for a, is defined by setting $r(a)_q$ equal to 1 and $r(a)_m$ equal to 0 for $m = 1, \ldots, n$, $m \neq q$. Another $n-$dimensional vector $o(a) = \{o(a)_m\}$, $m = 1, \ldots, n$, called the actual output for a, is defined by setting $o(a)_m$ equal to the output of the m^{th} neuron in the output layer (neuron with label $nq - n + m$) for each m, $m = 1, \ldots, n$. The total squared error between the desired responses $r(a)$ and the actual outputs $o(a)$, a in A, is then

$$E(w) = 1/2 \sum_{a \in A} |r(a) - o(a)|^2 = 1/2 \sum_{a \in A} \sum_{m=1}^{n} (r(a)_m - o(a)_m)^2,$$

where w is the unique vector in weight space corresponding to the current set of weights in the network. As E is implicitly defined in terms of the activation functions assigned to the neurons in the network, E has partial derivatives of all orders at any w. The training of the neural network on sample patterns in A is then accomplished in a batch learning manner by adjusting the weights between layers in the network using differential calculus so that a minimum to the error E as a function of the weights is obtained. With batch learning all weights are adjusted only after all patterns in A are processed as opposed to on-line learning with which all weights are adjusted each time a pattern in A is processed.

In this paper we discuss NEURBT, a Fortran 77 program for computing neural networks for classification using batch learning. NEURBT is based on Møller's scaled conjugate gradient algorithm [7] which is a variation of the traditional conjugate gradient method [5], better suited for the nonquadratic nature of neural networks. In what follows an outline of Møller's algorithm is presented that closely resembles the implementation of the algorithm in program NEURBT. In addition, other aspects of the implementation are discussed such as the efficient computation of gradients and multiplication of vectors by Hessian matrices that take place in Møller's algorithm, and the stochastic (re)initialization of weights. A copy of NEURBT can be obtained from `http://math.nist.gov/~JBernal`

2 Scaled Conjugate Gradient Algorithm

Program NEURBT is based on Møller's scaled conjugate gradient algorithm [7] for minimizing the total squared error E as a function of weights. Møller's algorithm, an outline of which is presented below, is based on the well-known conjugate gradient method [5] which works well for quadratic or nearly-quadratic functions. Since the Hessian matrix $E''(w)$ of the squared error function E at w may not be positive definitive for w in certain areas of weight space, Møller modified the conjugate gradient method based on the approach of the Levenberg-Marquardt algorithm [2]. If at some point during the execution of the conjugate gradient method for some p and w in $nw-$dimensional Euclidean space $\delta = p^t E''(w)p$ is computed resulting in a nonpositive δ, one makes δ positive by adding $p^t \lambda p$ to it for some $\lambda > 0$, i.e., by scaling the Hessian matrix $E''(w)$ with the appropriate $\lambda > 0$ so that δ becomes $p^t(E''(w) + \lambda I)p$, I the identity matrix. Once λ is initialized it is used and adjusted appropriately throughout the execution of the algorithm so that each δ computed as above remains positive. However, since the accuracy of the conjugate gradient method depends on approximating $E(w)$ with a quadratic function that involves $E''(w)$, care must be taken that the scaled $E''(w)$ does not produce a bad approximation. This is again taken care of by appropriately raising and lowering λ. The outline of the scaled conjugate gradient algorithm below includes the manipulations for raising and lowering λ. Here the column vector $E'(w)$ is the gradient of E at weight vector w. The outline closely resembles the implementation of Møller's algorithm in program NEURBT.

1. Stochastically initialize weight vector w_0,
   $k = 0$, $\epsilon_1 = 10^{-6}$, $\epsilon_2 = 10^{-4}$, $\lambda_0 = \epsilon_2$, $\bar{\lambda}_0 = 0$, $r_0 = p_0 = -E'(w_0)$,
   $success = true$.

2. Calculate second-order information: $s_k = E''(w_k)p_k$, $\delta_k = p_k^t s_k$.

3. Scale Hessian matrix: $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|p_k|^2$.

4. If $\delta_k \leq 0$ then scale Hessian matrix to make it positive definite:
   $\bar{\lambda}_k = 2(\lambda_k - \delta_k/|p_k|^2)$, $\delta_k = -\delta_k + \lambda_k|p_k|^2$, $\lambda_k = \bar{\lambda}_k$.

5. Calculate the step size: $\mu_k = p_k^t r_k$, $\alpha_k = \mu_k/\delta_k$.

6. Calculate the comparison parameter Δ_k: $\bar{w}_{k+1} = w_k + \alpha_k p_k$,
   $E\_q = E(w_k) + E'(w_k)^t \alpha_k p_k + 1/2(\alpha_k p_k^t E''(w_k)\alpha_k p_k + \lambda_k|\alpha_k p_k|^2)$,
   $\Delta_k = [E(w_k) - E(\bar{w}_{k+1})]/[E(w_k) - E\_q] = 2\delta_k[E(w_k) - E(\bar{w}_{k+1})]/\mu_k^2$.

7. Test for error reduction:
   If $\Delta_k \geq 0$ then a successful error reduction can be made:
   $w_{k+1} = \bar{w}_{k+1}$, $r_{k+1} = -E'(w_{k+1})$.

If $|r_{k+1}| < \epsilon_1$ then terminate and return w_{k+1} as desired minimum.
If $success = false$ or k mod $nw = 0$ then restart: $p_{k+1} = r_{k+1}$, $\lambda_{k+1} = \epsilon_2$, $\bar{\lambda}_{k+1} = 0$, $k = k + 1$, $success = true$, and go to step 2.
Else (if $success = true$ and k mod $nw \neq 0$) then create new conjugate direction: $\beta_k = (|r_{k+1}|^2 - r^t_{k+1} r_k)/\mu_k$, $p_{k+1} = r_{k+1} + \beta_k p_k$.
If $\Delta_k \geq 0.75$ then reduce the scale parameter: $\lambda_k = 1/2\lambda_k$.
Else (if $\Delta_k < 0$) error reduction is not possible:
$\bar{\lambda}_k = \lambda_k$, $success = false$.

8. If $\Delta_k < 0.25$ then increase the scale parameter: $\lambda_k = 4\lambda_k$.

9. if $success = false$ then go to step 3.
   Else set $\bar{\lambda}_{k+1} = 0$, $\lambda_{k+1} = \lambda_k$, $k = k + 1$, and go to step 2.

3 Computing the Gradient

In order to attempt to minimize the error E as a function of w using the scaled conjugate gradient algorithm as described above, the capability must exist for the efficient computation of the gradient $E'(w)$ of E at w and multiplication of a vector by the Hessian matrix $E''(w)$ of E at w. In this section we develop the formulas used in program NEURBT for the computation of the gradient $E'(w)$ as presented in [4].

Given $a \in A$, w in weight space, the error at w due to a is $E_a(w) = 1/2\sum_{m=1}^{n}(r(a)_m - o(a)_m)^2$. Thus, $E(w) = \sum_{a \in A} E_a(w)$. Writing w as $\{w_k\}$, $k = 1, \ldots, nw$, it follows that $\partial E/\partial w_k = \sum_{a \in A} \partial E_a/\partial w_k$ for each k, $k = 1, \ldots, nw$. Therefore, by fixing a in A, in what follows it will suffice to develop only the formulas associated with E_a.

As it will become apparent from the formulas below the calculations of the partial derivatives of E_a with these formulas must take place in a specific order, from right to left in the network. This is because each calculation corresponding to a given weight depends on calculations corresponding to other weights in the network to the right of the given weight. Computing in this manner is called backpropagation. However, it is an implementation issue that is taken care of in program NEURBT and not necessary for the development of the formulas.

Consider layers K, M, L, consecutive layers in the network from left to right, $\{y_k\}$, the set of outputs of neurons in layer K, $\{y_m\}$, the set of outputs of neurons in layer M, and $\{x_l\}$, the set of inputs of computing neurons in layer L. In particular consider y_j, the output of some neuron in layer K, and x_i, y_i, the input and output, respectively, of some computing neuron in layer M. In addition, for each y_k as above let w_{ki} be the weight such that $x_i = \sum_k w_{ki} y_k$; and for each y_m and x_l as above let w_{ml} be the weight such that $x_l = \sum_m w_{ml} y_m$.

Case 1. Layer K is not the input layer and layer M is a hidden layer so that layer L is either a hidden layer or the output layer.

Using the chain rule repeatedly we then get

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i}\frac{\partial x_i}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i}\frac{\partial(\sum_k w_{ki} y_k)}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i}\frac{\partial(w_{ji} y_j)}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} y_j.$$

$$\frac{\partial E_a}{\partial x_i} = \frac{\partial E_a}{\partial y_i}\frac{\partial y_i}{\partial x_i} = \frac{\partial E_a}{\partial y_i}\sigma'(x_i).$$

$$\frac{\partial E_a}{\partial y_i} = \sum_l \frac{\partial E_a}{\partial x_l}\frac{\partial x_l}{\partial y_i} = \sum_l \frac{\partial E_a}{\partial x_l}\frac{\partial(\sum_m w_{ml} y_m)}{\partial y_i} = \sum_l \frac{\partial E_a}{\partial x_l}\frac{\partial(w_{il} y_i)}{\partial y_i} =$$

$$\sum_l \frac{\partial E_a}{\partial x_l} w_{il}.$$

Thus,

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} y_j = \frac{\partial E_a}{\partial y_i}\sigma'(x_i) y_j = (\sum_l \frac{\partial E_a}{\partial x_l} w_{il})\sigma'(x_i) y_j.$$

Case 2. Layer K is the input layer so that layer M is the first layer in the network. Then $\{y_k\}$ can be replaced by $\{a_k\}$, the set of coordinates of input pattern a.

Thus, $x_i = \sum_k w_{ki} a_k$, and

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} a_j.$$

Once again

$$\frac{\partial E_a}{\partial x_i} = \frac{\partial E_a}{\partial y_i}\sigma'(x_i)$$

and

$$\frac{\partial E_a}{\partial y_i} = \sum_l \frac{\partial E_a}{\partial x_l} w_{il}.$$

Thus,

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} a_j = \frac{\partial E_a}{\partial y_i} \sigma'(x_i) a_j = (\sum_l \frac{\partial E_a}{\partial x_l} w_{il}) \sigma'(x_i) a_j.$$

Case 3. Layer M is the output layer of the network so that there is no layer L.

Then once again

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} y_j$$

and

$$\frac{\partial E_a}{\partial x_i} = \frac{\partial E_a}{\partial y_i} \sigma'(x_i).$$

With $\{y_m\}$ ordered so that for $m = 1, \ldots, n$, $y_m = o(a)_m$ then $E_a(w) = 1/2 \sum_{m=1}^n (r(a)_m - o(a)_m)^2 = 1/2 \sum_{m=1}^n (r(a)_m - y_m)^2$, so that

$$\frac{\partial E_a}{\partial y_i} = y_i - r(a)_i$$

and

$$\frac{\partial E_a}{\partial w_{ji}} = \frac{\partial E_a}{\partial x_i} y_j = \frac{\partial E_a}{\partial y_i} \sigma'(x_i) y_j = (y_i - r(a)_i) \sigma'(x_i) y_j.$$

Note that in all cases $\sigma'(x_i) = y_i(1 - y_i)$.

4 Fast Exact Multiplication by the Hessian

In this section we develop the formulas used in the implementation of the scaled conjugate gradient algorithm in NEURBT for the fast exact computation of the product of the Hessian matrix $E''(w)$ with an $nw-$dimensional vector v in the context of Møller's algorithm. With these formulas the calculation of the complete Hessian matrix is avoided. These formulas were originally derived by Pearlmutter [8] and involve the so-called $\mathcal{R}\{\cdot\}$ operator. As in the case of the gradient $E'(w)$, by fixing a in A, in what follows it will suffice to develop only the formulas associated with E_a. These formulas depend on the formulas developed above for the computation of the gradient $E'(w)$, thus simultaneously as $E'(w)$ is computed with backpropagation, the exact product of v and $E''(w)$ is computed with these formulas in either a feed-forward fashion or in the manner of backpropagation. But again this is an implementation issue that is taken care of in program NEURBT and not necessary for the development of the formulas.

Let f be a differentiable function from $nw-$dimensional Euclidean space into any other finite-dimensional Euclidean space. The $\mathcal{R}_v\{\cdot\}$ operator or simply the $\mathcal{R}\{\cdot\}$ operator is defined by

$$\mathcal{R}_v\{f(w)\} \equiv \frac{d}{dr} f(w+rv)\bigg|_{r=0} = f'(w)v,$$

where $f'(w)$ is the Jacobian matrix of f at w. In particular $\mathcal{R}_v\{E'(w)\} = E''(w)v$. Writing w as $\{w_k\}$, $k = 1, \ldots, nw$, it also follows that for each k, $k = 1, \ldots, nw$, $\mathcal{R}_v\{\partial E/\partial w_k\}$ is the k^{th} component of $E''(w)v$.

Given g, a differentiable function with domain and range appropriately defined, c a real number, then some equations involving $\mathcal{R}\{\cdot\}$ are satisfied:

$$\begin{aligned}
\mathcal{R}\{cf(w)\} &= c\mathcal{R}\{f(w)\} \\
\mathcal{R}\{f(w)+g(w)\} &= \mathcal{R}\{f(w)\} + \mathcal{R}\{g(w)\} \\
\mathcal{R}\{f(w)g(w)\} &= \mathcal{R}\{f(w)\}g(w) + f(w)\mathcal{R}\{g(w)\} \\
\mathcal{R}\{g(f(w))\} &= g'(f(w))\mathcal{R}\{f(w)\} \\
\mathcal{R}\{\frac{d}{dt}f(w)\} &= \frac{d}{dt}\mathcal{R}\{f(w)\} \\
\mathcal{R}\{w\} &= v.
\end{aligned}$$

With these equations and the formulas obtained in the previous section for the components of $E'_a(w)$, the formulas for the components of $E''_a(w)v$ can be derived. In what follows weights are doubly indexed. Since there is a one-to-one correspondence between the components of w and v then the components of v will be similarly indexed.

As in the previous section, consider layers K, M, L, consecutive layers in the network from left to right, $\{y_k\}$, the set of outputs of neurons in layer K, $\{y_m\}$, the set of outputs of neurons in layer M, and $\{x_l\}$, the set of inputs of computing neurons in layer L. In particular consider y_j, the output of some neuron in layer K, and x_i, y_i, the input and output, respectively, of some computing neuron in layer M. In addition, for each y_k as above let w_{ki} be the weight such that $x_i = \sum_k w_{ki} y_k$; and for each y_m and x_l as above let w_{ml} be the weight such that $x_l = \sum_m w_{ml} y_m$.

Case 1. Layer K is not the input layer and layer M is a hidden layer so that layer L is either a hidden layer or the output layer.

Applying $\mathcal{R}\{\cdot\}$ on x_i and y_i, we get the feed-forward formulas:

$$\mathcal{R}\{x_i\} = \mathcal{R}\{\sum_k w_{ki}y_k\} = \sum_k \mathcal{R}\{w_{ki}y_k\} = \sum_k(\mathcal{R}\{w_{ki}\}y_k + w_{ki}\mathcal{R}\{y_k\}) =$$

$$\sum_k(v_{ki}y_k + w_{ki}\mathcal{R}\{y_k\}).$$

$$\mathcal{R}\{y_i\} = \mathcal{R}\{\sigma(x_i)\} = \sigma'(x_i)\mathcal{R}\{x_i\}.$$

Applying $\mathcal{R}\{\cdot\}$ on $\partial E_a/\partial w_{ji}$, $\partial E_a/\partial x_i$, $\partial E_a/\partial y_i$, as computed in the previous section for case 1, we get the backpropagation formulas:

$$\mathcal{R}\{\frac{\partial E_a}{\partial w_{ji}}\} = \mathcal{R}\{\frac{\partial E_a}{\partial x_i}y_j\} = \mathcal{R}\{\frac{\partial E_a}{\partial x_i}\}y_j + \frac{\partial E_a}{\partial x_i}\mathcal{R}\{y_j\}.$$

$$\mathcal{R}\{\frac{\partial E_a}{\partial x_i}\} = \mathcal{R}\{\frac{\partial E_a}{\partial y_i}\sigma'(x_i)\} = \mathcal{R}\{\frac{\partial E_a}{\partial y_i}\}\sigma'(x_i) + \frac{\partial E_a}{\partial y_i}\mathcal{R}\{\sigma'(x_i)\} =$$

$$\mathcal{R}\{\frac{\partial E_a}{\partial y_i}\}\sigma'(x_i) + \frac{\partial E_a}{\partial y_i}\sigma''(x_i)\mathcal{R}\{x_i\}.$$

$$\mathcal{R}\{\frac{\partial E_a}{\partial y_i}\} = \mathcal{R}\{\sum_l \frac{\partial E_a}{\partial x_l}w_{il}\} = \sum_l \mathcal{R}\{\frac{\partial E_a}{\partial x_l}w_{il}\} =$$

$$\sum_l(\mathcal{R}\{\frac{\partial E_a}{\partial x_l}\}w_{il} + \frac{\partial E_a}{\partial x_l}\mathcal{R}\{w_{il}\} = \sum_l(\mathcal{R}\{\frac{\partial E_a}{\partial x_l}\}w_{il} + \frac{\partial E_a}{\partial x_l}v_{il}).$$

Case 2. Layer K is the input layer so that layer M is the first layer in the network. Then $\{y_k\}$ can be replaced by $\{a_k\}$, the set of coordinates of input pattern a.

Applying $\mathcal{R}\{\cdot\}$ on x_i and $\partial E_a/\partial w_{ji}$, as computed in the previous section for case 2, we get

$$\mathcal{R}\{x_i\} = \mathcal{R}\{\sum_k w_{ki}a_k\} = \sum_k \mathcal{R}\{w_{ki}a_k\} = \sum_k \mathcal{R}\{w_{ki}\}a_k = \sum_k v_{ki}a_k,$$

and

$$\mathcal{R}\{\frac{\partial E_a}{\partial w_{ji}}\} = \mathcal{R}\{\frac{\partial E_a}{\partial x_i}a_j\} = \mathcal{R}\{\frac{\partial E_a}{\partial x_i}\}a_j,$$

with the other formulas derived for case 1 above remaining the same.

Case 3. Layer M is the output layer of the network so that there is no layer L.

Applying $\mathcal{R}\{\cdot\}$ on $\partial E_a/\partial y_i$, as computed in the previous section for case 3, we get

$$\mathcal{R}\{\frac{\partial E_a}{\partial y_i}\} = \mathcal{R}\{y_i - r(a)_i\} = \mathcal{R}\{y_i\},$$

with the other formulas derived for case 1 above remaining the same.

Note that in all cases $\sigma''(x_i) = (1 - 2y_i)\sigma'(x_i)$, and $\sigma'(x_i) = y_i(1 - y_i)$.

5 Stochastic (Re)Initialization of Weights

Program NEURBT has the capability of restarting the scaled conjugate gradient algorithm each time it detects insufficient progress in the training of the sample patterns. Based mostly on ideas in [6], the following stochastic procedure is used to move away from the current position or solution in weight space (the last weight vector found with the scaled conjugate gradient algorithm) by randomly perturbing it several times as described below to obtain weight vectors in hopes that some are improvements over the current solution. These weight vectors are obtained by randomly perturbing the current solution inside a ball (using the maximum norm in weight space) with center at the current solution and a relatively large radius, a radius which is gradually reduced. If at a given radius, some of the weight vectors obtained are improvements, the best one is then used as the center around which the next random perturbations for the next (reduced) radius will take place. Otherwise the current center continues to be used as such. Once all pertubations are done, if there have been any improvements, the weight vector that is the best improvement is used as the initial solution for the next training with the scaled conjugate gradient algorithm. Otherwise, if none of the weight vectors obtained through perturbations is an improvement, one is randomly selected as the new initial solution for the next training even though it is worse than the current solution. The procedure follows.

1. Set K_1, K_2 to appropriate positive integers (e.g., $K_1 = 100$, $K_2 = 5$), $a = 2.0$, $k_2 = 0$, $rx = 2.0$, $flag = 1$.

2. If the scaled conjugate gradient algorithm has not been executed yet, i.e., it is the start of the execution of NEURBT, then set weight vector w_c to the zero vector in weight space, $w_m = w_c$, $E_m = \infty$.
   Else set w_c to the current solution of the scaled conjugate gradient algorithm, $w_m = w_c$, $E_m = E(w_m)$.

```
3. Set $k_1 = 0$, $k_2 = k_2 + 1$, $a = 0.5 \cdot a$, $w_c = w_m$.

4. Set $k_1 = k_1 + 1$.
   Generate weight vector $w_n$ of components random numbers in $[-a, a]$.
   Set $w = w_c + w_n$.
   If $flag = 2$ then
   if $E(w) < E_m$ then set $w_m = w$, $E_m = E(w_m)$;
   go to step 5.
   Else $(flag \neq 2)$
   if $E(w) < E_m$ then set $w_m = w$, $E_m = E(w_m)$, $flag = 2$, go to step 5;
   else $(E(w) \geq E_m)$ generate random number $rn$ in $[0, 1]$,
   if $rn < rx$ then set $w_t = w$, $rx = rn$.

5. If $k_1 \leq K_1$ go to step 4.

6. If $k_2 \leq K_2$ go to step 3.

7. If $flag = 2$ then set $w_0 = w_m$.
   Else $(flag \neq 2)$ set $w_0 = w_t$.
   Use $w_0$ as the initial solution for the next execution of the scaled conjugate gradient algorithm.
```

6 Numerical Results

6.1 Cushing Syndrome Classification

Here we present results from program NEURBT on a small example associated with the so-called Cushing syndrome. This is an example used in [3] as an application of neural networks for classification.

The Cushing syndrome is a disorder that occurs when the body is exposed to high levels of the hormone cortisol for a long time. Three types of the syndrome are recognized: adenoma, bilateral hyperplasia, and carcinoma. In the presence of the Cushing syndrome the following observations were made that represent urinary excretion rates (mg/24hr) of the steroid metabolites tetrahydrocortisone (in the second column below) and pregnanetriol (in the third column). Each line of observations has a label that appears in the first column, and each of the lines corresponds to an individual identified with each of the observations in the line, an individual with a known type of the syndrome. Accordingly, lines labeled a1, ..., a6 correspond to individuals with the adenoma type; lines labeled b1, ..., b10 correspond to individuals with the bilateral hyperplasia type; and lines labeled c1, ..., c5 correspond to individuals with the carcinoma type. Lines labeled u1, ..., u6 correspond to individuals with the syndrome, each individual with an unknown type of the syndrome. Finally, the fourth and fifth columns of the data below have the same data as the second and third columns, respectively, but on a log scale.

a1	3.1	11.70	1.1314021	2.45958884
a2	3.0	1.30	1.0986123	0.26236426
a3	1.9	0.10	0.6418539	-2.30258509
a4	3.8	0.04	1.3350011	-3.21887582
a5	4.1	1.10	1.4109870	0.09531018
a6	1.9	0.40	0.6418539	-0.91629073
b1	8.3	1.00	2.1162555	0.00000000
b2	3.8	0.20	1.3350011	-1.60943791
b3	3.9	0.60	1.3609766	-0.51082562
b4	7.8	1.20	2.0541237	0.18232156
b5	9.1	0.60	2.2082744	-0.51082562
b6	15.4	3.60	2.7343675	1.28093385
b7	7.7	1.60	2.0412203	0.47000363
b8	6.5	0.40	1.8718022	-0.91629073
b9	5.7	0.40	1.7404662	-0.91629073
b10	13.6	1.60	2.6100698	0.47000363
c1	10.2	6.40	2.3223877	1.85629799
c2	9.2	7.90	2.2192035	2.06686276
c3	9.6	3.10	2.2617631	1.13140211
c4	53.8	2.50	3.9852735	0.91629073
c5	15.8	7.60	2.7600099	2.02814825
u1	5.1	0.40	1.6292405	-0.9162907
u2	12.9	5.00	2.5572273	1.6094379
u3	13.0	0.80	2.5649494	-0.2231436
u4	2.6	0.10	0.9555114	-2.3025851
u5	30.0	0.10	3.4011974	-2.3025851
u6	20.5	0.80	3.0204249	-0.2231436

Log scale data above for observations in lines a1, ..., a6, b1, ..., b10, c1, ..., c5, was used as training data for NEURBT, and once training was completed on a 4−layer network associated with the data, log scale data for observations in lines u1, ..., u6, together with the trained network was used to identify with NEURBT the type (adenoma, bilateral hyperplasia, or carcinoma) corresponding to each of these lines. The classification results from the execution of NEURBT follow for each line of unknown type. Here the first columm of numbers contains outputs from the ouput node of the neural network corresponding to the ademona type; the second column contains outputs from the output node corresponding to the bilateral hyperplasia type; and finally the third column contains outputs from the output node corresponding to the carcinoma type.

u1	3.24477751E-05	0.999754658	2.89550184E-05
u2	0.00229908955	0.00168873688	0.99732707
u3	6.43877753E-06	0.999842854	0.000214464156
u4	0.993283221	0.016642477	5.17611453E-10
u5	9.02998701E-06	0.999543222	0.000649654443
u6	1.69426598E-05	0.997179771	0.00402886012

From these results it appears that adenoma is the type corresponding to line u4; bilateral hyperplasia is the type corresponding to lines u1, u3, u5, u6; and carcinoma is the type corresponding to line u2. This classification of these lines is consistent with the classification of the same lines in [3].

6.2 Wine Classification

Data in [1] is the result of a chemical analysis of wines produced in the same region in Italy from three different cultivars. Each line in the data corresponds to a wine and contains quantities of 13 constituents in the wine that were determined through the chemical analysis.

The 13 constituents were:
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids

8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Training data for program NEURBT was obtained from [1] as follows. The first 50 lines of data for wine from the first cultivar were extracted from the data and identified as Class 1 training data; the first 60 lines of data for wine from the second cultivar were extracted from the data and identified as Class 2 training data; and the first 40 lines of data for wine from the third cultivar were extracted from the data and indentified as Class 3 training data. In all cases each line of data consisted of 13 numbers corresponding in the same order to the quantities of constituents listed above. For example, the first line in the Class 1 training data appeared exactly as follows:

4.23,1.71,2.43,15.6,127,2.8,3.06,.28,2.29,5.64,1.04,3.92,1065

Using this data, NEURBT was then executed to train a 4−layer network associated with the data. For the purpose of testing the trained network the remaining 9 lines of data for wine from the first cultivar were extracted from the data and identified as Class 1 independent data; the remaining 11 lines of data for wine from the second cultivar were extracted from the data and identified as Class 2 independent data; and the remaining 8 lines of data for wine from the third cultivar were extracted from the data and indentified as Class 3 independent data.

The classification results from the execution of NEURBT follow for each line of independent data. Here the first columm of numbers contains outputs from the ouput node of the neural network corresponding to wine from the first cultivar; the second column contains outputs from the output node corresponding to wine from the second cultivar; and finally the third column contains outputs from the output node corresponding to wine from the third cultivar. The first 9 lines correspond to the 9 lines in the Class 1 independent data in the same order; the next 11 lines correspond to the 11 lines in the Class 2 independent data in the same order; and the final 8 lines correspond to the 8 lines in the Class 3 independent data in the same order.

```
0.997898696 1.55894382E-05 0.000718329838
0.997899514 1.56570618E-05 0.000714938256
0.997899491 1.5655163E-05 0.000715033071
0.997822451 9.83061807E-06 0.00118815891
0.9978991 1.56227882E-05 0.000716653417
0.997898055 1.565808E-05 0.000716251607
0.9978995 1.56561285E-05 0.000714986862
0.997897986 1.55308256E-05 0.000721294412
0.997899439 1.56508964E-05 0.000715246206
0.0325602964 0.968719073 0.00263130889
0.0324685933 0.968772038 0.00263806428
0.0324681146 0.968772314 0.00263809963
0.0324681071 0.968772309 0.00263810103
0.0324703628 0.968770996 0.00263793516
0.0324683634 0.968772171 0.00263808125
0.0324681146 0.968772314 0.00263809963
0.0324681146 0.968772314 0.00263809963
0.0324681142 0.968772314 0.00263809969
0.0324681146 0.968772314 0.00263809963
0.0324595001 0.968764647 0.00263979697
0.00210771733 0.00323003576 0.997239526
0.00210756517 0.00322870149 0.997240864
0.00210756559 0.00322870516 0.997240861
0.74630825 6.64272561E-05 0.480445691
0.00210756568 0.00322870595 0.99724086
0.00210763918 0.00322858059 0.997240827
0.00210756512 0.00322870101 0.997240865
0.00210756528 0.00322870247 0.997240863
```

From these results it appears that only the wine corresponding to the 4th line in the Class 3 independent data was classified incorrectly. Additional output from NEURBT confirms this:

```
Independent patterns classification results:
Class = 1 Total = 9 Correct = 9 Percentage = 100.
Class = 2 Total = 11 Correct = 11 Percentage = 100.
Class = 3 Total = 8 Correct = 7 Percentage = 87.5
```

7 Summary

NEURBT, a Fortran 77 program for computing neural networks for classification using batch learning, was discussed. Since program NEURBT is based on Møller's scaled conjugate gradient algorithm which is a variation of the traditional conjugate gradient method, better suited for the nonquadratic nature of neural networks, an outline of Møller's algorithm was presented that resembles its implementation in program NEURBT. Important aspects of the implementation were discussed such as the efficient computation of gradients and multiplication of vectors by Hessian matrices that are required by Møller's algorithm. Accordingly, formulas for the product of vectors by Hessian matrices depending on those for the gradients used in NEURBT were developed. Because of this dependence it was pointed out that calculations with these formulas of the gradient at a vector and the product of the Hessian at the same vector with another vector in the context of Møller's algorithm occur simultaneously and take place in either a feed-forward fashion or in the manner of backpropagation. Finally, another aspect of the implementation was discussed which is the stochastic (re)initialization of weights, the main purpose of which is to restart the scaled conjugate gradient algorithm each time NEURBT detects insufficient progress in the training of the sample patterns. With this capability NEURBT has a better chance of reaching a global optimal solution. However if at some point during its execution it detects that it has either climbed the wrong mountain or gotten stuck in a valley one too many times it will stop while producing what it considers to be the best current solution. A copy of NEURBT can be obtained from `http://math.nist.gov/~JBernal`

References

[1] Bache, K., Lichman, M.: UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml (2013).

[2] Fletcher, R.: Practical Methods of Optimization. John Wiley & Sons (1975).

[3] Gongwer, G., Iyer, M., Kong, M.: Stat 6601 Classification: Neural Networks V&R 12.2. www.sci.csueastbay.edu/~jkwon/classes/stat_6601 /PROJECT/6601_presentation_Classification_NeuralNetwork.ppt (2010).

[4] Gonzalez, R. C., Woods, R. E.: Digital Image Processing. Pearson Prentice Hall (2008).

[5] Hestenes, M. R., Stiefel, E. L.: Methods of Conjugate Gradients for Solving Linear Systems. Journal of Research of the National Bureau of Standards 49(6) (1952) 409–436.

[6] Masters, T.: Practical Neural Network Recipes in C++. Academic Press (1993).

[7] Møller, M. F.: A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. Neural Networks 6(4) (1993) 525–533.

[8] Pearlmutter, B. A.: Fast Exact Multiplication by the Hessian. Neural Computation 6(1) (1994) 147–160.