Back propagation algorithm

Brian Fiedler

November 18, 2001

Abstract

The backpropagation algorithm coded in `nn.f` is briefly summarized. The notation follows that of Chapter 11 in *Neural Network Design* by Hagan, Demuth and Beale.

1 The neural network (the forward model)

A vector of predictors $\mathbf{p}$ serves as the input vector to the forward model. The forward model progresses through M layers, from layer $m = 1$ to $m = M$. The output from layer m will be a vector $\mathbf{a}^m$. This output vector either serves as either the input vector to a subsequent layer, or, if it is $\mathbf{a}^M$, it is the predictand of the model. The length of $\mathbf{a}^m$ is denoted as N^m.

To aid with the notation of the algorithm, the input vector is also denoted as $\mathbf{a}^0$. The action of layer m can be described in two steps. First, there is a linear transformation of the input vector $\mathbf{a}^{m-1}$ with the multiplication by a weight matrix $\mathbf{W}^m$ and the addition of a bias vector $\mathbf{b}^m$. Second, each element of the resultant vector $\mathbf{n}^m$ is operated on by a transfer function f^m, which produces the elements of the output vector $\mathbf{a}^m$. Therefore, for $i = 1$ to N^m,

$$n_i^m = \sum_{j=1}^{N^{m-1}} w_{ij}^m a_j^{m-1} + b_i^m \tag{1}$$

$$a_i^m = f^m\left(n_i^m\right) \ . \tag{2}$$

A neural network will usually have at least one nonlinear transfer function, with the logsig function being a common choice:

$$f(x) = \frac{1}{1+e^{-cx}} \ , \tag{3}$$

where c is a value to be chosen. Common advice in the use of neural networks is to scale the predictors to have zero mean and a variance near unity. With such scaling, $c = 1$ is a workable choice in (3).

One of the most popular neural networks is a two-layer model, with a nonlinear transfer function in only the first layer. For the second layer, $f^2(x) = x$.

Optimal values for the weight matrices and bias vectors are achieved with the use of the use of *training data*, which consists of many pairs of *input vectors* $\mathbf{p}$ and *target vectors* $\mathbf{t}$. For each of the pairs, the error in the prediction is calculated as

$$\widehat{F} = \left(\mathbf{a}^M - \mathbf{t}\right)^2 . \tag{4}$$

The sensitivity of this error to changes in each element of every $\mathbf{W}^m$ and $\mathbf{b}^m$ is then calculated using the chain rule of differential calculus. Based on the sensitivity analysis, small adjustments are then made to the weight matrices and bias vectors. The most error reduction with the least "disruption" of the existing element occurs if the adjustments are proportional to the sensitivity.

The goal of training is to reduce the average error over all the pairs of input and target vectors. There is little purpose in dwelling on the optimal adjustment of the weight matrices and bias vectors for a given pair of input vector and target vector; the same weight matrices and bias vectors will eventually have to be used for all input vectors. The adjustment process is kept simple: a small nudge is given with the use of each data pair. With many passes through the training data set, which consists of many pairs of input and target vectors, the accumulation of nudges to the weight matrices and bias vectors will also tend to reduce the average error. This iterative procedure is the essence of training a neural network. The craft of training usually consists of calculating an average error for the entire training data set each after epoch of training (an epoch is a pass through the entire data set). A portion of data pairs is often set aside as a *verification data set*, with which no training is done, but for which an average error is also calculated. Monitoring the average error of both the training data set and the verification data set will may provide a *stopping critereon* for the training, which is often when the error with verification data set either begins to rise, or rise substantially relative to the training data. Such a rise is often indicative of "overfitting" or "fitting to noise", meaning fitting to relations that are not universal, but exist only in the training data set.

2 Reducing the error

Here is shown the method to adjust w_{ij}^m and b_i^m in the effort to reduce $\widehat{F}$. "Minimization" is attempted with a simple *method of steepest descent*. For each set of $(\mathbf{p}, \mathbf{t})$, adjustment is done with:

$$w_{ij}^m = w_{ij}^m - \alpha \frac{\partial \widehat{F}}{\partial w_{ij}^m} \tag{5}$$

$$b_i^m = b_i^m - \alpha \frac{\partial \widehat{F}}{\partial b_i^m} . \tag{6}$$

α is sometimes referred to as the learning rate. Small values of α cause slow convergence and increase the computational expense. Large values of α may overemphasize adjustment with the current data pair, and global minimization may suffer. Large values of α may also cause instability and lack of convergence. An acceptable value of

α may be determined experimentally. A common practice is to reduce α during the training, which allows for initial speed of convergence, but ultimately a training of a neural network that is optimized over all the training data.

Note that with the use of unscaled, dimensional data, the mthod of steepest descent is not dimensionally homogeneous. Unscaled data, with very different numerical ranges, is known to cause difficulty in the convergence of the minimization using the method of steepest descent. However, the method of steepest descent will work satisfactorily with scaled variables.

The gradient in the error $\widehat{F}$ is found with

$$s_i^m \equiv \frac{\partial \widehat{F}}{\partial n_i^m}. \tag{7}$$

In (5) and (6) we need

$$\begin{aligned} \frac{\partial \widehat{F}}{\partial w_{ij}^m} &= \frac{\partial \widehat{F}}{\partial n_i^m}\frac{\partial n_i^m}{\partial w_{ij}^m} && (8)\\ &= \frac{\partial \widehat{F}}{\partial n_i^m} a_j^{m-1} && (9)\\ &= s_i^m a_j^{m-1}. && (10) \end{aligned}$$

and

$$\begin{aligned} \frac{\partial \widehat{F}}{\partial b_i^m} &= \frac{\partial \widehat{F}}{\partial n_i^m}\frac{\partial n_i^m}{\partial b_i^m} && (11)\\ &= \frac{\partial \widehat{F}}{\partial n_i^m} && (12)\\ &= s_i^m. && (13) \end{aligned}$$

But how do we find s_i^m?

3 Backpropagating the sensitivities

$$\begin{aligned} s_i^m &\equiv \frac{\partial \widehat{F}}{\partial n_i^m} && (14)\\ &= \sum_{j=1}^{N} \frac{\partial n_j^{m+1}}{\partial n_i^m}\frac{\partial \widehat{F}}{\partial n_j^{m+1}} && (15)\\ &= \sum_{j=1}^{N} \frac{\partial n_j^{m+1}}{\partial n_i^m} s_j^{m+1}. && (16)\\ & && (17) \end{aligned}$$

But,

$$\begin{aligned}\frac{\partial n_j^{m+1}}{\partial n_i^m} &= \sum_{k=1}^{N} \frac{\partial n_j^{m+1}}{\partial a_k^m} \frac{\partial a_k^m}{\partial n_i^m} && (18)\\ &= \frac{\partial n_j^{m+1}}{\partial a_i^m} \frac{\partial a_i^m}{\partial n_i^m} && (19)\\ &= w_{ji}^{m+1} \frac{\partial f\left(n_i^m\right)}{\partial n_i^m}. && (20)\end{aligned}$$

So,

$$s_i^m = \frac{\partial f^m\left(n_i^m\right)}{\partial n_i^m} \sum_{j=1}^{N} s_j^{m+1} w_{ji}^{m+1}. \tag{21}$$

Back propagation is started with s_i^M, obtained with:

$$\begin{aligned}s_i^M &= -2\left(t_i - a_i^M\right) \frac{\partial a_i^M}{\partial n_i^M} && (22)\\ &= -2\left(t_i - a_i^M\right) \frac{\partial f^M\left(n_i^M\right)}{\partial n_i^M}. && (23)\end{aligned}$$