Optimization of a Neural Network

William S. Harlan

Feb 1999

Introduction

Many discussions of neural networks unnecessarily introduce a large vocabulary of specialized terms. In fact, neural networks are a simple system of recursive equations that can be optimized by conventional means. Any good book on optimization provides all the necessary tools [1]. We need only unambiguously write down the equations we are using, identify the objective function, and calculate a few derivatives.

The Recursive Equations

Neural-network equations map vectors of known data $\mathbf{d}^m$ to corresponding known vectors of parameters $\mathbf{p}^m$, indexed by m. The vector of parameters might be actual physical properties or statistical probabilities. Data can be physical measurements or numbers that index possible events. The equations contain unknown coefficients, or weights, that the user adjusts until known parameters are predicted accurately. Then it is hoped that unknown parameters can be estimated from new measurements of data.

Let's look at how simple these equations are. Intermediate vectors $\mathbf{x}^{m,n}$ are expressed recursively as a function of other vectors $\mathbf{x}^{m,n-1}$, usually with a different dimensionality. The first vector $\mathbf{x}^{m,0}$ corresponds to data $\mathbf{d}^m$, and the final vector $\mathbf{x}^{m,N}$ corresponds to the parameters $\mathbf{p}^m$ to be estimated. First, a pre-established non-linear transform $\mathbf{f}^n$ is applied to each element of a vector. Then linear transforms $\underset{\sim}{\mathbf{W}}^n$ calculate each sample of the output vector as a weighted sum of samples from the input vector.

$$x_i^{m,n} = \sum_j W_{ij}^n f_j^n(\mathbf{x}^{m,n-1}), \tag{1}$$

$$\text{or } \mathbf{x}^{m,n} = \underset{\sim}{\mathbf{W}}^n \cdot \mathbf{f}^n(\mathbf{x}^{m,n-1}) \text{ (in vector notation).} \tag{2}$$

$$\text{Find parameters } \mathbf{p}^m \approx \mathbf{x}^{m,N} \tag{3}$$

$$\text{from data } \mathbf{d}^m = \mathbf{x}^{m,0}. \tag{4}$$

Usually the non-linear functions $\mathbf{f}^n$ apply independently to each element of the vector. Functions are expected to be monotonic over the range $0 \le f(x) < f(x') \le 1$ for values $0 \le x < x' \le 1$. The derivative $df(x)/dx$ should also be continuous. Usually, the shape is a

sigmoid or linear. Because the notation presents no difficulty, I write this non-linear scalar function as a more general vector function.

I could also have reversed the order of linear and non-linear operators. A reversed order is equivalent to setting $\mathbf{f}^1$ and $\underset{\sim}{\mathbf{W}}^N$ to identity operations. This form (1) is somewhat easier to manipulate.

This nonlinear recursion can be abbreviated as

$$\mathbf{p}^m \approx \mathbf{x}^{m,N}(\mathbf{d}^m, \underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N) = \mathbf{x}^{m,N}. \tag{5}$$

Perturbations of an Objective Function

The best weights $\underset{\sim}{\mathbf{W}}^n$ are assumed to minimize the following least-squares objective function F, summed over all known pairs m of data measurements $\mathbf{d}^m$ and model parameters $\mathbf{p}^m$.

$$\min_{\{W_{ij}^n\}} F(\underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N) = \sum_{m,k} [p_k^m - x_k^{m,N}(\mathbf{d}^m, \underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N)]^2 = \sum_m \|\mathbf{p}^m - \mathbf{x}^{m,N}\|^2. \tag{6}$$

A perturbation of the weights $\delta\underset{\sim}{\mathbf{W}}^n$ will result in a linearized perturbation δF of the objective function:

$$\delta F(\underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N) = -\sum_{m,k} (p_k^m - x_k^{m,N}) \delta x_k^{m,N} = -\sum_m (\mathbf{p}^m - \mathbf{x}^{m,N}) \cdot \delta \mathbf{x}^{m,N}, \tag{7}$$

$$\text{where } \delta \mathbf{x}^{m,0} = \mathbf{0} \tag{8}$$

$$\text{and } \delta x_i^{m,n} = \sum_j \delta W_{ij}^n \cdot f_j^n(\mathbf{x}^{m,n-1}) + \sum_j W_{ij}^n \sum_k \frac{\partial}{\partial x_k} f_j^n(\mathbf{x}^{m,n-1}) \delta x_k^{m,n-1}, \tag{9}$$

$$\text{or } \delta \mathbf{x}^{m,n} = \delta \underset{\sim}{\mathbf{W}}^n \cdot \mathbf{f}^n(\mathbf{x}^{m,n-1}) + \underset{\sim}{\mathbf{W}}^n \cdot \underset{\sim}{\nabla} \mathbf{f}^n(\mathbf{x}^{m,n-1}) \cdot \delta \mathbf{x}^{m,n-1}. \tag{10}$$

Unperturbed variables retain their original reference values from the original non-linear forward transform (1).

The perturbations (9) of the vector elements $\delta \mathbf{x}^{m,n}$ are expressed as a linear function of perturbed weights $\delta \underset{\sim}{\mathbf{W}}^n$. We can abbreviate the recursive equations (9) as a single linear transform $\underset{\sim}{\mathbf{G}}^{m,n}$.

$$\delta x_k^{m,N} = \sum_{n,i,j} G_{kij}^{m,n} \delta W_{ij}^n \tag{11}$$

$$\text{or } \delta \mathbf{x}^{m,N} = \sum_n \underset{\sim}{\mathbf{G}}^{m,n} : \delta \underset{\sim}{\mathbf{W}}^n. \tag{12}$$

Gradient optimization also requires the adjoint of this linearization. If the linearized forward transform is expressed as a matrix, then the adjoint transform is just the transpose of this matrix. A matrix would be unwieldy, but the recursive version of the adjoint equations is not.

$$\delta W_{ij}^n = \sum_m \delta x_i^{m,n} f_j^n(\mathbf{x}^{m,n-1}), \tag{13}$$

$$\text{or } \delta \underset{\sim}{\mathbf{W}}^n = \sum_m \delta \mathbf{x}^{m,n} [\mathbf{f}^n(\mathbf{x}^{m,n-1})]^* \quad \text{(outer product)}, \tag{14}$$

$$\text{where } \delta \mathbf{x}^{m,N} = -(\mathbf{p}^m - \mathbf{x}^{m,N}) \delta F \tag{15}$$

$$\text{and } \delta x_k^{m,n-1} = \sum_j \frac{\partial}{\partial x_k} f_j^n(\mathbf{x}^{m,n-1}) \sum_i W_{ij}^n \delta x_i^{m,n}, \tag{16}$$

$$\text{or } \delta \mathbf{x}^{m,n-1} = [\underset{\sim}{\nabla} \mathbf{f}^n(\mathbf{x}^{m,n-1})]^* \cdot (\underset{\sim}{\mathbf{W}}^n)^* \cdot \delta \mathbf{x}^{m,n}. \tag{17}$$

These perturbations do not equal those of the forward linearization. The adjoint recursion can also be abbreviated with the linear transform $\underset{\sim}{\mathbf{G}}^{m,n}$.

$$\delta W_{ij}^n = \sum_{m,k} G_{kij}^{m,n} \delta x_k^{m,N}, \tag{18}$$

$$\text{or } \delta \underset{\sim}{\mathbf{W}}^n = (\underset{\sim}{\mathbf{G}}^{m,n})^* \cdot \delta \mathbf{x}^{m,N}. \tag{19}$$

If the linear transform $\underset{\sim}{\mathbf{G}}^{m,n}$ were written as a single large matrix, then indeed the matrix would be identical in the forward and adjoint equations (11) and (18). For optimization, fortunately, we need not construct this matrix explicitly (as would be required by singular-value decomposition). Instead, we need only be able to multiply this linear transform or its adjoint by specific perturbations, using the recursions (9) and (16).

Optimization

The simplest neural network "training" algorithm adjusts the previous choice of weights by a scaled gradient. This recursive algorithm is called back-propagation.

1. Initialize each weight matrix $\underset{\sim}{\mathbf{W}}^n$.
2. Calculate $\Delta W_{ij}^n = \sum_{m,k} G_{kij}^{m,n} [p_k^m - x_k^{m,N}(\mathbf{d}^m, \underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N)]$,
   $\Delta \underset{\sim}{\mathbf{W}}^n = (\underset{\sim}{\mathbf{G}}^{m,n})^* \cdot (\mathbf{p}^m - \mathbf{x}^{m,N})$.
3. Replace each $\underset{\sim}{\mathbf{W}}^n$ by $\underset{\sim}{\mathbf{W}}^n + \epsilon \Delta \underset{\sim}{\mathbf{W}}^n$.
4. Return to step 2.

To reduce the objective function, the perturbation reverses the sign of the gradient. The small scale factor ϵ is sometimes fixed *a priori* and never changed. If the scale factor is too small, then many consecutive steps may move in the same direction. If the scale factor is too large, the perturbations may increase rather than decrease the objective function and may even diverge. Although this algorithm is most commonly cited, we can easily do better. Let us turn to standard non-linear optimization methods.

Many steepest-descent algorithms would replace step 3 by a line search to find an optimum scale factor.

1. Initialize each weight matrix $\underset{\sim}{\mathbf{W}}^n$.

2. Calculate $\Delta \underset{\sim}{\mathbf{W}}^n$ as before.
3. Find α to minimize $\sum_{m,k} [p_k^m - x_k^{m,N}(\mathbf{d}^m, \underset{\sim}{\mathbf{W}}^1 + \alpha\Delta\underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N + \alpha\Delta\underset{\sim}{\mathbf{W}}^N)]^2$
4. Replace each $\underset{\sim}{\mathbf{W}}^n$ by $\underset{\sim}{\mathbf{W}}^n + \alpha\Delta\underset{\sim}{\mathbf{W}}^n$.
5. Return to step 2.

This revised algorithm is guaranteed to find a local minimum where the gradient is zero. Step sizes are large. The line search can use combinations of parabolic fitting and golden sections for speed and robustness.

A good estimate of the scale factor can be estimated without an explicit line search.

1. Initialize each $\underset{\sim}{\mathbf{W}}^n$.
2. Calculate $\Delta \underset{\sim}{\mathbf{W}}^n$ as before.
3. Calculate $\Delta\mathbf{x}^{m,N} = \sum_n \underset{\sim}{\mathbf{G}}^{m,n} : \Delta\underset{\sim}{\mathbf{W}}^n$.
4. Find α to minimize $\sum_{m,k} [p_k^m - \alpha\Delta x_k^{m,N}]^2$,
   or equivalently $\alpha = (\sum_{m,k} p_k^m \Delta x_k^{m,N}) / \sum_{m,k} \Delta x_k^{m,N} \Delta x_k^{m,N})$.
5. Replace each $\underset{\sim}{\mathbf{W}}^n$ by $\underset{\sim}{\mathbf{W}}^n + \alpha\Delta\underset{\sim}{\mathbf{W}}^n$.
6. Return to step 2.

When the linearization is a good approximation (particularly in the vicinity of a minimum), the scale factor will be very accurate. Each iteration will be much more efficient than a line search. If the scaled perturbation increases the objective function in early non-linear iterations, then a line search can be used until the linearization improves.

The convergence of these steepest descent algorithms should be improved by a PARTAN or Fletcher-Reeves algorithm, which retains and combines consecutive gradients. Neural network references use the word "momentum" to describe similar strategies, usually with invariable scale factors.

An even more efficient alternative is to solve the linearized least-squares problem fully in each iteration — a Gauss-Newton approach:

1. Initialize each $\underset{\sim}{\mathbf{W}}^n$.
2. Find the perturbations $\{\underset{\sim}{\mathbf{w}}^1, \ldots, \underset{\sim}{\mathbf{w}}^N\}$ that minimize
   $\sum_{m,k} [p_k^m - x_k^{m,N}(\mathbf{d}^m, \underset{\sim}{\mathbf{W}}^1, \ldots, \underset{\sim}{\mathbf{W}}^N) - \sum_{n,i,j} G_{kij}^{m,n} w_{ij}^n]^2$.
3. Replace each $\underset{\sim}{\mathbf{W}}^n$ by $\underset{\sim}{\mathbf{W}}^n + \underset{\sim}{\mathbf{w}}^n$.
4. Return to step 2.

Since the objective function in step 2 is a fully quadratic function of the perturbations $\mathbf{w}^n$, we can use iterative least-squares methods like conjugate gradients.

If a steepest descent algorithm were applied with an unlimited number of infinitesimal perturbations, then the changing weights would continuously decrease the objective function until a local minimum was reached. Visualize following a path that is directly downhill at every point. The path cannot ascend, no matter what may be beyond the next rise. Practical descent algorithms take large step sizes and may step over a small local increase in the objective function, to find another minimum beyond.

To increase the chances of finding the global minimum with the pseudo-quadratic objective function (6), we could initialize with different initial weights and discard suboptimum local minima. If different weights consistently lead to the same solution, then the objective function is probably convex and has only one global minimum.

References

[1] David G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison Wesley, 1973.