CISC 371 Class 17

Back-Propagating Scale Factors of Gradient Components

Article: Rumelhart et al, *Learning internal representations by back-propagating errors* [1]

Main Concepts:

- *Forward evaluation: compute objective value*
- *Back-propagate: scale factors*
- *Back-propagation: compute gradient components*

Sample Problem, Machine Learning: How can we efficiently compute descent for a neural network?

The fundamental algorithm for neural networks is called *back propagation*, which is widely attributed to Rumelhart *et al* . [1] but which has much earlier sources.

In this course, we have posed the problem of training a neural network as a problem in unconstrained optimization. For a weight vector $\vec{w}$ that gathers the weights of individual neurons, and an objective function $f(\vec{w})$ that is to be minimized, our solution was to use steepest descent with a fixed stepsize. Using a "learning rate" of η, an iterative solution is

$$\vec{w}_{k+1} = \vec{w}_k - \eta \left[\underline{\nabla} f(\vec{w}_k)\right]^T$$

In Class 15, we derived an objective function for a simple multi-layer neural network that used a squared-error objective. The large matrices in our linear-algebra formulation were avoided by back-propagating scale factors and finding the descent vectors from the inputs to the neurons.

We will work through this faster process of back-propagation by using a simple example. In Section 15.4, we proposed a neural network that had two layers of neurons: a single output neuron and two "hidden" neurons. The inputs to this network were size-2 observations that represented points in a plane. The data were arranged in an "exclusive-or" pattern, with observations in the odd quadrants labeled as $+1$ and in the even quadrants as 0. A two-layer network with two inputs and a single output is shown in Figure 17.1. The output, which is Layer 3, has the weights vector ${}_3\vec{w}$ of Equation 15.2. The "hidden" neurons, which are in Layer 2, have weights ${}_2\vec{w}_1$ and ${}_2\vec{w}_2$ that we gathered in the matrix ${}_2W$ of Equation 15.3.

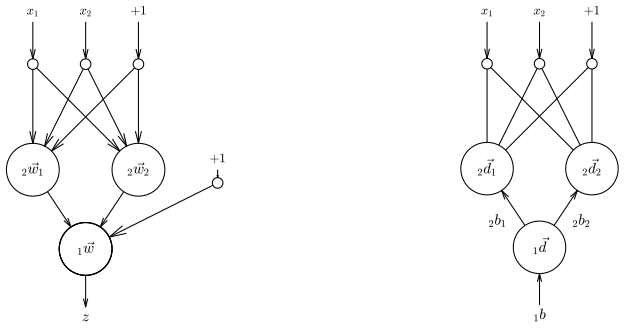

Figure 17.1: A neural net with a single hidden layer. (A) A simplified network with size-2 inputs and two neurons in the hidden layer; the neuron labels are their respective weight vectors. (B) In back-propagation, the neuron labels are the scale factors for the gradient components and connector labels are scale factors for the neurons.

For the purposes of creating a scalar objective function that has a single vector argument, we gathered these terms as

$$\begin{aligned} {}_2W &\stackrel{\text{def}}{=} \begin{bmatrix} {}_2\vec{w}_1 & {}_1\vec{w}_2 \end{bmatrix} \\ {}_2\vec{w} &\stackrel{\text{def}}{=} \begin{bmatrix} {}_2\vec{w}_1 \\ {}_2\vec{w}_2 \end{bmatrix} \\ \vec{w} &\stackrel{\text{def}}{=} \begin{bmatrix} {}_3\vec{w} \\ {}_2\vec{w} \end{bmatrix} \end{aligned} \tag{17.1}$$

The objective function for the weight vector of Equation 17.1 depends on an observation $\underline{x}$. The linear terms for Layer 2 are the product of the observation $\underline{x}$ and the weights ${}_2W$; these are the arguments to a generally nonlinear activation function ${}_2\phi$. These outputs are augmented with a constant $+1$ and then are the inputs to the neuron of Layer 3, which performs the same kind of computation with a possibly different activation function ${}_3\phi$. The "feed-forward" computations are

$$\begin{aligned} {}_2\underline{u} &= [\underline{x}]\, {}_2W \\ {}_2\vec{\phi} &= \begin{bmatrix} {}_2\phi({}_2u_1) \\ {}_2\phi({}_2u_2) \end{bmatrix} \\ {}_3\underline{x} &= \begin{bmatrix} {}_2\vec{\phi}^T & 1 \end{bmatrix} \\ {}_3u &= [{}_3\underline{x}]\, {}_3\vec{w} \\ z(\vec{w}) &= {}_3\phi({}_3u) \end{aligned} \tag{17.2}$$

The output $z(\vec{w})$ of Equation 17.2, and a label y of the observation $\underline{x}$, are used to create an objective function. For a squared-error objective, the residual r and the objective function for this 3-layer network are

$$\begin{aligned} r &= y - z(\vec{w}) \\ f_3(\vec{w}) &= r^2 \end{aligned} \tag{17.3}$$

The gradient, which we found by applying the Chain Rule to Equation 17.3, is presented in Equation 15.12.

17.1 Back-Propagation of Scale Factors

The descent vector of our simplified neural network is found by applying the Chain Rule to the objective function of Equation 17.3. In practice, we use the back-propagation algorithm described in Section 15.2 of Class 15.

The computation of the descent vector can be conceptually represented as a sequence. Using ${}_lb$ to represent the back-propagated scale factor at Level l, the computations for our simple network can be written as:

1. Compute the scale factor of the objective as
$$ {}_3b = \frac{\partial f_3}{\partial r}\frac{\partial\, r}{\partial\, {}_3\phi} = -r(\vec{w})$$
2. Compute the Layer 3 gradient component, and the descent vector for the objective, as
$$\begin{aligned} {}_3\vec{s} &= {}_3\vec{x} \\ {}_3\vec{d} &= -{}_3\vec{s}\; {}_3\psi\; {}_3b \\ \vec{d} &\leftarrow {}_3\vec{d} \end{aligned}$$
3. Compute the scale factor for Layer 2 as
$$ {}_2\vec{b} = {}_3\vec{w}_{1\ldots 2}({}_3\psi_3 b)$$
4. Compute the Layer 2 gradient components as
$$\begin{aligned} {}_2S &= {}_2\vec{x}[{}_2\vec{\psi} \odot {}_2\vec{b}]^T \\ {}_2\vec{s} &= \text{vec}\,({}_2S) \\ {}_2\vec{d} &= -{}_2\vec{s} \end{aligned}$$

5. Compute the descent vector as

$$\vec{d} \leftarrow \begin{bmatrix} \vec{d} \\ {}_2\vec{d} \end{bmatrix}$$

Observe that, in Step 3, we have *back-propagated* the scale factor from the objective. We are "sending", to Layer 2, the objective scale factor times the local derivative; the latter is the input to Layer 3, which is ${}_3\vec{x}$.

Observe that, in Step 4, we have used this back-propagated scale factor. The "local" descent vector, ${}_2\vec{d}$, depends on: the inputs to Layer 2, which are ${}_2\vec{x}$; the "local" derivatives of the activation functions, which are ${}_2\vec{\phi}$; and the back-propagated scale factor, which are ${}_2\vec{b}$.

For our simple network, we can write the back-propagation terms in Step 3 individually as

$$\begin{array}{rcl} {}_2b_1 & = & {}_3w_1 \; {}_3\psi \; {}_3b \\ {}_2b_2 & = & {}_3w_2 \; {}_3\psi \; {}_3b \end{array} \tag{17.4}$$

Likewise, we can write the steepest-vector-propagation terms in Step 4 individually as

$$\begin{array}{rcl} {}_2s_1 & = & {}_2x_1 \; {}_2\psi_1 \; {}_2b_1 \\ {}_2s_2 & = & {}_2x_2 \; {}_2\psi_2 \; {}_2b_2 \end{array} \tag{17.5}$$

The combination of Equation 17.4 and Equation 17.5 have the same effects as the vector computations presented above.

17.2 Example: Forward Evaluation

Consider a specific example of our simplified neural network. Suppose that the weight vector, separated into the components for the neurons in the layers, is

$${}_3\vec{w} = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.0 \end{bmatrix} \qquad {}_2W = \begin{bmatrix} {}_2\vec{w}_1 & {}_2\vec{w}_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 1.0 \\ 0.5 & 1.0 \\ 0.0 & 1.0 \end{bmatrix} \tag{17.6}$$

Suppose further that the observation $\underline{x}$ and its label y are

$$\underline{x} = \begin{bmatrix} 0.5 & 1.0 & 1 \end{bmatrix} \qquad y = 1 \tag{17.7}$$

For simplicity, we will use the logistic activation function for all three neurons in the network. The activation function and its derivative are

$$\begin{aligned} \phi(t) &\stackrel{\text{def}}{=} \frac{1}{1+e^{-t}} \\ \psi(t) &= \phi'(t) \\ &= \phi(t)(1-\phi(t)) \end{aligned} \tag{17.8}$$

We will first perform computations on the network of Figure 17.1, using the values in Equation 17.6 and Equation 17.7, by a forward evaluation. The values, corresponding to Equation 17.2 plus the derivatives and objective function, are

$$\begin{aligned} {}_2\underline{u} &= \begin{bmatrix} 0.75 & 2.5 \end{bmatrix} \\ {}_2\underline{\phi} &= \begin{bmatrix} 0.67918 & 0.92414 \end{bmatrix} \\ {}_2\underline{\psi} &= \begin{bmatrix} 0.21789 & 0.070104 \end{bmatrix} \\ {}_3\underline{x} &= \begin{bmatrix} 0.67918 & 0.92414 & 1 \end{bmatrix} \\ {}_3u &= -0.12248 \\ {}_3\phi &= 0.46942 \\ {}_3\psi &= 0.24906 \\ r &= 0.53058 \\ f(\vec{w}) &= 0.14076 \end{aligned} \tag{17.9}$$

In the calculations of Equation 17.9, some observations include:

- Linear terms ${}_2\underline{u}$ in Layer 2 are of the magnitude of unity, or ± 1
- Activation terms ${}_2\underline{\phi}$ in Layer 2 are of the magnitude of unity
- Derivative terms ${}_2\underline{\psi}$ in Layer 2 are an order of magnitude less than the activation terms
- Activation term ${}_3\underline{\phi}$ in Layer 3 is of the magnitude of unity
- Residual term r is of the magnitude of unity

In this example, for weights that are of the order of magnitude of unity, to neural network preserves to order of magnitude as the input $\underline{x}$ is used to compute the residual term. The objective function will be bounded between 0 and 1 because is is the square of the residual term.

17.3 Example: Back-Propagation

Using the values in Equation 17.6, Equation 17.7, and Equation 17.9, we can perform the back-propagation computations. The scale factors and gradient 1-forms that we find using back-propagation are:

Step	Name	Symbol	Value
1.0	Objective scale:	${}_3b$	-0.53058
2.0	Layer 3 descent:	${}_3\vec{d}$	$\begin{bmatrix} +0.089753 \\ +0.12212 \\ +0.13215 \end{bmatrix}$
3.0	Layer 2 scale:	${}_2\vec{b}$	$\begin{bmatrix} -0.066075 \\ +0.066075 \end{bmatrix}$
4.1	Layer 2 steepest:	${}_2S$	$\begin{bmatrix} -0.0071987 & +0.0023160 \\ -0.0143973 & +0.0046321 \\ -0.0143973 & +0.0046321 \end{bmatrix}$
4.3	Layer 2 descent:	${}_2\vec{d}$	$\begin{bmatrix} +0.0071987 \\ +0.0143973 \\ +0.0143973 \\ -0.0023160 \\ -0.0046321 \\ -0.0046321 \end{bmatrix}$

In these calculations of back-propagation of scale factors, some observations include:

- The output scale factor ${}_3b$ is on the order of unity
- The output descent vector ${}_3\vec{d}$ is an order of magnitude less than unity
- The hidden-layer scale factors ${}_2\vec{b}$ are two orders of magnitude less than unity
- The hidden-layer descent vectors ${}_2\vec{d}_j$ are three orders of magnitude less than unity

We can see that the derivative terms $\psi(t) = \phi'(t)$ have tended to be numerically small. The effect, in this example, is that different layers train at different rates.

References

[1] Rumelhart DE, Hinton GE, Williams RJ: Learning representations by back-propagating errors. *Nature* 323(6088):533–536, 1986