Neural Network Fitting

Stephen Vardeman

Analytics Iowa LLC

ISU Statistics and IMSE

Back-propagation and partials of the *k*-th output

The most common fitting algorithm for neural networks is the "back-propagation algorithm" or the "delta rule." It is simply a gradient descent algorithm for the entire set of weights involved in making the outputs (in the toy example used in these slides the αs and βs). Rather than completely detail such an algorithm for even the toy example, we will here only lay out the heart of what is needed.

For a training set of size N, loss $L\left(\hat{\mathbf{f}}\left(\mathbf{x}_i\right), y_i\right)$ incurred for input case i when the K predictions $\hat{f}_k\left(\mathbf{x}_i\right)$ are made (corresponding to the K output nodes), and where the sum of such losses is to be minimized, if one can find the partial derivatives of the coordinates of $\hat{\mathbf{f}}\left(\mathbf{x}\right)$ with respect to the weights the chain rule will give the partials of the total loss and allow iterative search in the direction of a negative gradient of the total loss. So we begin with description of how to find partials for $\hat{f}_k\left(\mathbf{x}\right)$, a coordinate of the fitted output vector.

Notation

Consider a neural network with layers of hidden nodes indexed by $h = 1, 2, \ldots, H$ beginning with the layer immediately before the output layer and proceeding (right to left in a diagram like the one used as an example) to the one that is built from linear combinations of the coordinates of $\mathbf{x}$. Use the notation m_h for the number of nodes in layer h, *including* a node representing the "bias" input 1 (represented by $x_0 = 1$ and $z_0 = 1$ in the toy example). For a real-valued activation function of a single real variable σ, define a vector-valued function $\boldsymbol{\sigma}_m : \Re^m \to \Re^m$ by

$$\boldsymbol{\sigma}_m (u_1, u_2, \ldots, u_m) = (\sigma(u_1), \sigma(u_2), \ldots, \sigma(u_m))$$

To reduce notational clutter, we will abuse notation somewhat and not subscript $\boldsymbol{\sigma}_m$, but rather write only $\boldsymbol{\sigma}$, leaving it to the reader to recall that $\boldsymbol{\sigma}$ outputs vectors of the same dimension as its argument. And it will be convenient to presume that both the input and output of a $\boldsymbol{\sigma}$ are *row* vectors.

Values produced at hidden nodes

For $\mathbf{A}^H$ a $(p+1) \times (m_H - 1)$ matrix of (weight) parameters, the relationship between the input $\mathbf{x}$ and vector of values (say $\mathbf{z}_H$) in the last hidden layer is

$$\mathbf{z}'_H = \left(1, \sigma\left(\left(1, \mathbf{x}'\right) \mathbf{A}^H\right)\right)$$

Next, for $\mathbf{A}^{H-1}$ an $m_H \times (m_{H-1} - 1)$ matrix of parameters the relationship between the vectors of values in the last and next to last hidden layers is

$$\mathbf{z}'_{H-1} = \left(1, \sigma\left(\mathbf{z}'_H \mathbf{A}^{H-1}\right)\right)$$

and so on to the $h = 1$ case of ($\mathbf{A}^h$ an $m_{h+1} \times (m_h - 1)$ matrix of parameters)

$$\mathbf{z}'_h = \left(1, \sigma\left(\mathbf{z}'_{h+1} \mathbf{A}^h\right)\right)$$

Forward and backward passes

Then for $\mathbf{A}^0$ an $m_1 \times K$ matrix of parameters and g_k a function of K real variables, the kth coordinate of the output is

$$g_k\left(\mathbf{z}_1'\mathbf{A}^0\right)$$

This series of relationships allows (via what is known as a "forward pass" through them) the computation of **z**s and predictions for a fixed set of coefficients collected in the **A**s and an input vector **x**.

Then partial derivatives of the kth coordinate of the response (at that input and set of coefficients) can be found via the "backward pass" based on the K partials of g_k, the derivative of σ, the recursions above, and the results of the forward pass. We illustrate this next.

A partial of a response wrt a weight

$$
\begin{aligned}
\frac{\partial \hat{y}_k}{\partial a^1_{ij}} &= \sum_{l=1}^{K} g_k^{(l)}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right)\frac{\partial}{\partial a^1_{ij}}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right)_l \\
&= \sum_{l=1}^{K} g_k^{(l)}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right)\frac{\partial}{\partial a^1_{ij}}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0_l\right) \\
&= \sum_{l=1}^{K} g_k^{(l)}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right)\sum_{k=1}^{K} a^0_{kl}\frac{\partial}{\partial a^1_{ij}}\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)_k \\
&= \sum_{l=1}^{K} g_k^{(l)}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right) a^0_{jl}\frac{\partial}{\partial a^1_{ij}}\sigma\left(\mathbf{z}_2'\mathbf{A}^1_j\right) \\
&= \sum_{l=1}^{K} g_k^{(l)}\left(\left(1, \sigma\left(\mathbf{z}_2'\mathbf{A}^1\right)\right)\mathbf{A}^0\right)\sigma'\left(\mathbf{z}_2'\mathbf{A}^1_j\right) a^0_{jl} z_{2i}
\end{aligned}
$$

"And so on" for other $\hat{y}_k$s and a^h_{ij}s.

Backward pass for partials in general

In general one is faced with the functional form for the kth coordinate of the output made by successive compositions using the activation function and linear combinations with coefficients in the matrices $\mathbf{A}^h$, from which partials $\dfrac{\partial \hat{y}_k}{\partial a_{ij}^h}$ are obtainable in the style above, by repeatedly using the chain rule. No doubt some appropriate use of vector calculus and corresponding notation could improve the looks of these expressions and recursions can be developed, but what is needed should be clear. Further, in some contexts numerical approximation of these partials may be the most direct and efficient means of obtaining them.

Partials of the loss

Then for loss $L(\hat{\mathbf{f}}, y)$ let

$$L_k(\hat{\mathbf{f}}, y) = \frac{\partial}{\partial \hat{f}_k} L(\hat{\mathbf{f}}, y)$$

For a an element of one of the $\mathbf{A}^h$ matrices, the partial derivative of the contribution of case i to a total loss with respect to it is

$$\sum_{k=1}^{K} L_k\left(\hat{\mathbf{f}}(\mathbf{x}_i), y_i\right) \frac{\partial}{\partial a} g_k\left(\mathbf{z}_1'(\mathbf{x}_i)\mathbf{A}^0\right)$$

(for $\mathbf{z}_1(\mathbf{x}_i)$ the set of values from the final hidden nodes and partials found as above) and the partial derivative of the total loss with respect to it is

$$D(a) = \sum_{i=1}^{N} \sum_{k=1}^{K} L_k\left(\hat{\mathbf{f}}(\mathbf{x}_i), y_i\right) \frac{\partial}{\partial a} g_k\left(\mathbf{z}_1'(\mathbf{x}_i)\mathbf{A}^0\right)$$

Gradient descent

The gradient of the total loss as a function of the matrices of weights then has entries $D(a)$ and an iterative search to optimize total loss with a current set of iterates a_{current} can produce new iterates

$$a_{\text{new}} = a_{\text{current}} - \gamma D\left(a_{\text{current}}\right)$$

for some "learning rate" $\gamma > 0$.

Losses and “stochastic” gradient descent

Of course, in SEL/univariate regression contexts, it is common to have $K = 1$ and take $L(\hat{f}, y) = (\hat{f} - y)^2$. In K-class classification models, it is most common to use K-dimensional output $\hat{\mathbf{g}} = \left(g_1\left(\mathbf{z}_1'\mathbf{A}^0\right), \ldots, g_K\left(\mathbf{z}_1'\mathbf{A}^0\right)\right)$ (with the "softmax" g_k) and to employ the cross-entropy loss

$$L(\hat{\mathbf{g}}, y) = -\sum_{k=1}^{K} I[y = k] \ln g_k(\mathbf{x})$$

There are several possibilities for regularizing the ill-posed fitting problem for neural nets. One possibility is to employ "stochastic gradient descent" and newly choose a random subset of the training set for use at each iteration of fitting. (In this regard, it is popular to even go so far as to employ only a single case at each iteration.)