Neural Network to Minimize Average Squared Error

Marshall Farrier

November 12, 2014

1 Problem

Create a neural network that outputs real values not restricted to the interval $[0, 1)$. Mean squared error should be the error measure, as it is obviously a hack just to expand the unit interval out to cover a wider range of real values.

2 Solution Idea

The idea is to use the normal equation to get the least-squared-error linear mapping from the last hidden layer to the prediction. Then use back propagation taking that matrix as a given. Since moving an appropriate amount in a direction determined by the matrix of partial derivatives (of weights with respect to error measure) will improve the error *given* the linear transformation at the end. So we should get double improvement, and hopefully fewer calculations for improved performance, on each step when on each iteration we use back propagation *given* the final linear transformation from the normal equation of the prior iteration, then calculate the new normal equation.

3 Implementation

First to work out the derivation on the simplest case, where we map each set of features to a single real value.

3.1 3-layer Neural Network (1 hidden layer)

The first problem is to calculate all the partial derivatives with respect to a weight matrix $\mathbf{w}$ with a neural network that processes feature matrix $\mathbf{X}$ in the following way:

$$\mathbf{X} \to \mathbf{X} \cdot \mathbf{w} \to f(\mathbf{X} \cdot \mathbf{w}) \to f(\mathbf{X} \cdot \mathbf{w}) \cdot \mathbf{v} \approx \mathbf{y} \tag{1}$$

where $\mathbf{X}$ is the matrix of features (including the constant feature) with 1 data point in each row; $\mathbf{w}$ is a matrix of weights by which $\mathbf{X}$ is multiplied before a nonlinear function f is applied element-wise to $\mathbf{X} \cdot \mathbf{w}$ in order to create the non-constant features in the hidden layer. f is what gives us the nonlinear distortion needed to approximate a nonlinear target function. Presumably we will want f to be something like the sigmoid function or $tanh$. They also have a fairly tight spread, which may or may not be a desirable quality. $\mathbf{v}$ is the vector (because we are only allowing 1 label per row) representing the mapping of the hidden layer to the output. For purposes of back propagation, the idea is to treat $\mathbf{v}$ as a constant. Also, for $\mathbf{v}$ in this context, we drop the first term, because the constant feature of the hidden layer doesn't vary with $\mathbf{w}$.

The goal is to choose values of $\mathbf{w}$ that minimize error where the error function is defined as:

$$z = \frac{1}{n} \parallel f(\mathbf{X} \cdot \mathbf{w}) \cdot \mathbf{v} - \mathbf{y} \parallel^2 \tag{2}$$

n is the number of rows in the dataset, and z is shorthand for the mean of the squared difference between predicted and actual label over the input data. We're looking for the partial derivatives over z with respect to each $w_{j,k}$ in the matrix $\mathbf{w}$.

3.1.1 Unrolling the Matrices

In detail, here is what we're trying to minimize:

$$z = \frac{1}{n} \sum_{i=1}^{n} ((\sum_{k=1}^{p} f(\sum_{j=0}^{m} x_{i,j} \cdot w_{j,k}) \cdot v_k) - y_i)^2 \tag{3}$$

The inconsistency in numbering between the rows (starting with 0) and columns (starting with 1) of $\mathbf{w}$ is because the $0th$ row is the factor applied to the constant components of $\mathbf{x}$. I start the column numbering at 1, however, because the *1st* column provides the coefficients for deriving the *1st* hidden feature, leaving room for the constant $0th$ hidden feature, for which we don't have to worry about derivatives with respect to $\mathbf{w}$ because the constant hidden feature doesn't change regardless of how we vary $\mathbf{w}$. The i index starts with 1 because it corresponds to rows in our data set, of there are n and not $n + 1$.

3.1.2 Partial Derivatives

We now need to find the matrix of partial derivatives $\frac{\partial z}{\partial w_{a,b}}$. Differentiating the above equation with respect to $w_{a,b}$ we get:

$$\frac{\partial z}{\partial w_{a,b}} = \frac{2}{n}\sum_{i=1}^{n}(f(\sum_{j=0}^{m} x_{i,j} \cdot w_{j,b}) \cdot v_b - y_i) \cdot (x_{i,a} \cdot v_b \cdot f'(\sum_{j=0}^{m} x_{i,j} \cdot w_{j,b})) \tag{4}$$

$$= \frac{2v_b}{n}\sum_{i=1}^{n} x_{i,a} \cdot f'(\sum_{j=0}^{m} x_{i,j} \cdot w_{j,b}) \cdot (v_b \cdot f(\sum_{j=0}^{m} x_{i,j} \cdot w_{j,b}) - y_i) \tag{5}$$

Note that index k occurs outside of the function f, so we can ignore that sum in the partial derivatives with respect to the elements of column k of $\mathbf{w}$. But we can't get rid of index j because that sum occurs inside the function f.

3.1.3 Choosing a specific f

The function f can in principle be anything, but we'll certainly want it to be nonlinear because it is what is going to allow us to approximate nonlinear target functions, and it has to be differentiable to allow regression. It's certainly worth exploring what functions might be generally optimal or more suited to particular types of learning problems. But to begin with, I'll work through the math using the sigmoid function:

$$S(t) = \frac{1}{1+e^{-t}} \tag{6}$$

It seems intuitively like it should be a good choice, but other choices, including unbounded functions, are certainly also worth exploring.

As its derivative we have:

$$S'(t) = \frac{e^{-t}}{(1+e^{-t})^2} \tag{7}$$

$$= \frac{1}{1+e^{-t}} \cdot \frac{e^{-t}}{1+e^{-t}} \tag{8}$$

$$= \frac{1}{1+e^{-t}} \cdot (1 - \frac{1}{1+e^{-t}}) \tag{9}$$

$$= S(t) \cdot (1 - S(t)) \tag{10}$$

Before plugging $S(t)$ and $S'(t)$ into our equation for $\frac{\partial z}{\partial w_{a,b}}$, I'm going to introduce a new variable $t_{i,b}$ to simplify our notation for the repeated term used as argument for f:

$$t_{i,b} = \sum_{j=0}^{m} x_{i,j} \cdot w_{j,b} \tag{11}$$

$t_{i,b}$ is actually just the $b-th$ hidden feature for the $i-th$ row of training data. So, regardless how we choose f, this sum has already been saved and won't have to be recalculated. Now, replacing the sum with our new variable $t_{i,b}$ and setting f and f' to S and S' respectively, we get:

$$\frac{\partial z}{\partial w_{a,b}} = \frac{2v_b}{n} \sum_{i=1}^{n} x_{i,a} \cdot S'(t_{i,b}) \cdot (v_b \cdot S(t_{i,b}) - y_i) \tag{12}$$

$$= \frac{2v_b}{n} \sum_{i=1}^{n} x_{i,a} \cdot S(t_{i,b}) \cdot (1 - S(t_{i,b})) \cdot (v_b \cdot S(t_{i,b}) - y_i) \tag{13}$$

To simplify as much as possible for a vectorized solution, we can introduce the matrix $\mathbf{T} = S(\mathbf{X} \cdot \mathbf{w})$. $\mathbf{T}$ is the matrix of coefficients $t_{j,k}$ and is identical to the hidden layer without the constant feature:

$$\frac{\partial z}{\partial \mathbf{w}} = \frac{2}{n} \cdot \mathbf{X}^{\intercal} \cdot (\mathbf{T} \odot (1 - \mathbf{T}) \odot ((\mathbf{T} \cdot \mathbf{v} - \mathbf{y}) \cdot \mathbf{v}^{\intercal})) \tag{14}$$

where $\odot$ represents element-wise matrix multiplication.

3.2 Multiple Hidden Layers

Forward and back propagation should work as usual except for the changes outlined above for using the least-squared-error linear mapping from the last hidden layer to the known labels.