Feedforward Neural Network Back Propagation

1 Overview

Neural networks (NN) are an effective tool for picking out subtleties in the correlations of our data. Initially, they may appear to be a bit of a black box. However, deconstructing their anatomy shows that all they really are is an excessive application of the chain rule. We will also show that the parameters in a neural network are input weights and biases that are fed into the respective activation functions. The weights, well, weight the inputs for that neuron. The bias is an additive term that ignores the inputs.

When the neural network is initialized, the weights are randomly selected. They need not add to one. We randomly select the weights to speed up the learning process. However, we set the bias term to have the initial value of one. Then we feed the first batch of data (recall the repeated cross-validation analogy) through all hidden layers and finally through the output layer (i.e. forward through the NN). We simply apply whatever the weights are, multiply them by the features (x-variables), and plug them into the activation function of the first layer. Then we plug this value as a "new" feature into the next hidden layer. Eventually we get to the output layer, which gives us our $\hat{y}$.

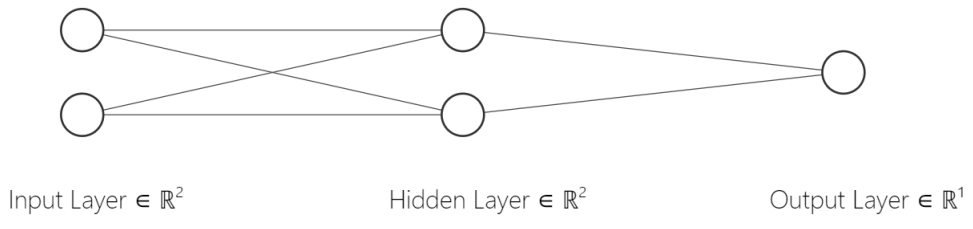
Once we have fed the data forward, we use back propagation to determine how much we need to adjust the parameters of the model (i.e. the weights and bias terms). At the end of the day, this is just an application of the chain rule. We have many nested functions. Each adjustment is based upon the learning rate for whichever gradient descent method we are using. Recall the adjustment takes the form

$$\hat{\theta}_{t+1} = \hat{\theta}_t - \eta \nabla_\theta L(X, y; \theta) \tag{1}$$

where $\hat{\theta}_t$ is a vector of the current weights and biases, η is the learning rate, $\nabla_\theta L(X, y; \theta)$ is a vector of the first derivative of the loss/cost function with respect to θ, and $\hat{\theta}_{t+1}$ is the next iteration of estimated weights and biases.

2 Setup

For the sake of exposition, we are going to use an over simplified neural network architecture as shown below:



We are going to use the MSE as the cost function, which you should be familiar with by this point in the semester. We will use the sigmoid activation function for the hidden and output neurons:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

Note that

$$\sigma'(z) = \sigma(z)(1 - (z)) \tag{3}$$

We are going to use mini-batch gradient descent where each batch has two observations. We will show one iteration of mini-batch gradient descent over two input features. Here is some made up data over made up countries:

Share in LF (y)	U-rate (%) (x_1)	GDP (%Δ) (x_2)
0.5	10	-5
0.6	3	2
y_3	$x_{1,3}$	$x_{2,3}$
$\vdots$	$\vdots$	$\vdots$

Note that the share of the population in the labor force is bounded between zero and one, as is our output layer activation function. Coincidence?

For the computations, we need to introduce some notation. We will use:

$$variable^{layer}_{neuron,index}$$

Let the hidden layer have value 1 and the output layer have value 2. In the hidden layer, let the upper neuron be 1 and the lower be 2. Let the output neuron be 1 (in this case, this would be redundant as there is only one output neuron. We can have more neurons in general).

To be concrete, the hidden layer bottom neuron activation function is $\sigma^1_2(z^1_2)$. The we can define:

$$z^1_2 = \sum_{i=1}^{2} x_i w^1_{2,i} + b^1_2 \tag{4}$$

Our **objective** is to show how we will adjust b^2_1 and $w^1_{2,1}$.

2.1 Initial Parameter Values

These are numbers that I randomly chose. Woo.

Layer	Neuron	Input 1 Weight	Input 2 Weight	Bias
Hidden	Top	$w^1_{1,1} = 0.2$	$w^1_{1,2} = 0.9$	$b^1_1 = 1$
Hidden	Bottom	$w^1_{2,1} = 0.7$	$w^1_{2,2} = 0.5$	$b^1_2 = 1$
Output	The only one	$w^2_{1,1} = 0.1$	$w^2_{1,2} = 0.5$	$b^2_1 = 1$

Note that if we had three neurons in our only hidden layer, the hidden layers would still only have two weights, but the output layer would gain an additional weight for a total of three weights and a bias.

Not sure where to put this, so let

$$\eta = 0.1$$

3 Solving the First Iteration

We are going to show by how much we will change b_1^2 and $w_{2,1}^1$ in the first batch of our (very) mini-batch gradient descent. We will first determine the slope for the first observation for both of these parameters. Let the observations be indexed by k. Because we are using the MSE, we have:

$$\frac{\partial C}{\partial \theta_0} = \frac{1}{K} \sum_{k=1}^{K} \frac{\partial C_k}{\partial \theta_0} \tag{5}$$

where $K = 2$ in our demonstration. We also have

$$C_k = (\hat{y}_k - y_k)^2 \tag{6}$$

Before we get going, we should solve for the inputs at each neuron.

INPUTS

$$\begin{aligned}
z_1^1 &= 0.2(10) + 0.9(-5) + 1 \\
&= -1.5 \\
\sigma_1^1 &= (z_1^1) \\
&\approx 0.18 \\
z_2^1 &= 0.7(10) + 0.5(-5) + 1 \\
&= 5.5 \\
\sigma_2^1 &= \sigma(z_2^1) \\
&\approx 0.99 \\
z_1^2 &= 0.1\sigma_1^1 + 0.5\sigma_2^1 + 1 \\
&\approx 1.52 \\
\sigma_1^2 &= \sigma(z_1^2) \\
&\approx 0.82 \\
&= \hat{y}_1
\end{aligned}$$

IMPORTANT: when you solve for this yourself, you should plug in the the nested activation functions to avoid rounding issues (as was done above).

3.1 b_1^2

We need to solve for the slope of our cost function given in equation 6 with respect to b_1^2; that is, the partial derivative. Observe that C_k is a function of σ_1^2, which is a function of z_1^2, which is a function of b_1^2. With all of these nested functions, you should be jumping for joy because you have realized we get to apply the chain rule a bunch! With a slight abuse of notation, we have:

$$\frac{\partial C_k}{\partial b_1^2} = \frac{\partial C_k(\sigma_1^2)}{\partial \sigma_1^2}\frac{\partial \sigma_1^2(z_1^2)}{\partial z_1^2}\frac{\partial z_1^2(b_1^2)}{\partial b_1^2} \tag{7}$$

All we have to do is solve for the three components on the right hand side of equation 7. We need to use equations 3, 4, and 6. Shall we?

$$\begin{aligned}
\frac{\partial C_1(\sigma_1^2)}{\partial \sigma_1^2} &= 2(\sigma_1^2 - y_k) \\
\frac{\partial \sigma_1^2(z_1^2)}{\partial z_1^2} &= \sigma(z_1^2)(1 - \sigma(z_1^2)) \\
\frac{\partial z_1^2(b_1^2)}{\partial b_1^2} &= 1
\end{aligned}$$

So, um, we can plug things in now.

$$\begin{aligned}
\frac{\partial C_1}{\partial b_1^2} &= \frac{\partial C_1(\sigma_1^2)}{\partial \sigma_1^2}\frac{\partial \sigma_1^2(z_1^2)}{\partial z_1^2}\frac{\partial z_1^2(b_1^2)}{\partial z_1^2} \\
&= 2(\sigma(z_1^2) - y_k)\sigma(z_1^2)(1 - \sigma(z_1^2))1 \\
&\approx 2(0.82 - 0.5)0.82(1 - 0.82) \\
&\approx 0.09
\end{aligned}$$

Tada! We just solved for the slope of the bias[1] for the single neuron in the output layer. Time to move on to solving for a weight in a hidden layer!

3.2 $w_{2,1}^1$

We now need to solve for the derivative of 6 with respect to $w_{2,1}^1$. All we have to do is apply the chain rule.

$$\frac{\partial C_1}{\partial w_{2,1}^1} = \frac{\partial C_1}{\partial \sigma_1^2}\frac{\partial \sigma_1^2}{\partial z_1^2}\frac{\partial z_1^2}{\partial \sigma_2^1}\frac{\partial \sigma_2^1}{\partial z_2^1}\frac{\partial z_2^1}{\partial w_{2,1}^1}$$

So,

$$\frac{\partial C_1}{\partial \sigma_1^2} = 2(\sigma_1^2 - y_1)$$

[1] If we were instead solving for a weight, say the first one, we would instead obtain $\frac{\partial z_1^2}{\partial w_{1,1}^2} = \sigma_1^1$. This means we would multiply by the value of σ_1^1 (≈ 0.18) as show in the inputs equations instead of 1.

$$\frac{\partial \sigma_1^2}{\partial z_1^2} = \sigma_1^2(1-\sigma_1^2)$$
$$\frac{\partial z_1^2}{\partial \sigma_2^1} = w_{1,2}^2$$
$$\frac{\partial \sigma_2^1}{\partial z_2^1} = \sigma(z_2^1)(1-\sigma(z_2^1))$$
$$\frac{\partial z_2^1}{\partial w_{2,1}^1} = x_1$$

which gives us

$$\begin{aligned}\frac{\partial C_1}{\partial w_{2,1}^1} &= 2(\sigma(z_1^2)-y_1)\sigma(z_1^2)(1-\sigma(z_1^2))w_{1,2}^2\sigma(z_2^1)(1-\sigma(z_2^1))x_1 \\ &\approx 2(0.82-0.5)0.82(1-0.82)0.5(0.99(1-0.99))10 \\ &\approx 0.002\end{aligned}$$

See why it is called the chain rule now?

4 First Batch Update

I will save you the time and tell you that $\frac{\partial C_2}{\partial \sigma_1^2} \approx 0.06$ and $\frac{\partial C_1}{\partial w_{2,1}^1} \approx 0.003$ We need to take the average of these values with the ones we solved for to determine the update to the parameters of interest. Using equation 1, we have

$$\theta - \eta\nabla_\theta C = \begin{bmatrix} w_{1,1}^1 \\ w_{1,2}^1 \\ b_1^1 \\ w_{2,1}^1 \\ w_{2,2}^1 \\ b_2^1 \\ w_{1,1}^2 \\ w_{1,2}^2 \\ b_1^2 \end{bmatrix} - 0.1 \begin{bmatrix} ? \\ ? \\ ? \\ \frac{1}{2}(0.002+0.003) \approx 0.0025 \\ ? \\ ? \\ ? \\ ? \\ \frac{1}{2}(0.09+0.06) \approx 0.080 \end{bmatrix}$$

So, we have after the first mini-(mini-mini-)batch that we are updating $w_{2,1}^1$ by -0.00025 and b_1^2 by -0.0075.[2] Welp, if you can replicate this with different numbers, then I think you have a good understanding of the anatomy of a neural network. Say, that sounds like a good homework problem!

[2]It just happens to be with these made up numbers that they are negative. I hope it is obvious that we can have positive changes to our initial values