Data Science 210 Final Project

Thomas Kwashnak

Fall 2021

.

1 Background

2 Design

During this section, a variety of variables and variable syntax will be used. The following is a list that explains each variable. Remember that for a given "layer", the output is considered the 0th layer, the hidden layer is considered the 1st layer, and the input is considered the 2nd layer.

$\vec{y}^n$	The output values of the nodes in the n^{th} hidden layer.
y_i^n	The output value of node i in the n^{th} hidden layer.
y	By default, if y is left alone, it represents the output value, known as y_0^0
$\hat{y}$	The expected final output
W^n	The weight matrix for a given layer n, as it applies to the values $\vec{y}^{n+1}$ (the outputs of the previous layer)
$w_{i,j}^n$	The weight value for the value passed from node j of layer $n+1$ as it is passed to node i of layer n

2.1 Derivations and Equations

The first part of designing the algorithms needed is to derive the math that will be needed in order to do every necessary function. Each of the following sub-sections describe how a given equation is derived.

2.1.1 Sigmoid Activation Function (σ)

In our neural network, we will use an activation function to condense the output of a node down to a [0-1] scale. In this lab, we will use a sigmoid function. Below is the sigmoid function and it's derivative.

$$\sigma(n) = \frac{1}{1+e^{-n}}$$

$$\sigma'(n) = \sigma(n) * (1 - \sigma(n))$$

When using this function on a matrix or vector, we simply just apply the sigma function for all individual values in the matrix/vector.

2.1.2 Forward Feeding Equation

This equation represents what the output will be for a given input $\vec{y^2}$. The basic principal is derived from the following relationship for the ouputs of layer n:

$$\vec{y^n} = \sigma(W_n * \vec{y^{n+1}})$$

We can chain this rule to find the output vector. Since the output vector is a 1x1 matrix, it will result in a scalar value

$$y = \vec{y^0} = \sigma(W_0 * \sigma(W_1 * \vec{y^2}))$$

2.1.3 The Loss Function

In order to measure the effectiveness of the function, we need to use a loss function that relates the output of the neural network to the expected result (from the data). We will use a simple loss function that takes the difference of these values, and then squares it to remove any negatives. We can use backwards propagation in order to change values to minimize this error, training the network to get our desired result.

$$L = (y_0^0 - \hat{y})^2 = (y - \hat{y})^2$$

2.1.4 Derivative of Layer-0 Weights

In back propagation, we need to find the derivative of each of the weights in terms of the loss function. First, we need to find the derivative of the loss function with respect to the weights that connect the hidden layer with the output layer. The derivative can be simplified using the chain rule as follows:

$$\frac{\delta L}{\delta w_{0,i}^0} = \frac{\delta L}{\delta y} * \frac{\delta y}{\delta w_{0,i}^0}$$

We can factor this out to be the following.
For simplifcation, $\sigma(\ldots) = \sigma(\sum_i (w_{0,i}^0 * \sigma(W^1 * \vec{y^2})_{i,0})) = y$.

$$\frac{\delta L}{\delta w_{0,i}^0} = 2(y - \hat{y}) * (\sigma'(\ldots)) * (\sigma((W^1 * \vec{y^2})_{i,0}))$$

$$\frac{\delta L}{\delta w_{0,i}^0} = 2(y - \hat{y}) * (\sigma(\ldots)(1 - \sigma(\ldots))) * (\sigma(\sum_h (w_{i,h}^1 * y_h^2))$$

$$\frac{\delta L}{\delta w_{0,i}^0} = 2(y - \hat{y}) * (y * (1 - y)) * (\sigma(\sum_h (w_{i,h}^1 y_h^2))$$

$$\frac{\delta L}{\delta w_{0,i}^0} = 2(y - \hat{y}) * (y - y^2) * (\sigma(\sum_h (w_{i,h}^1 * y_h^2))$$

And for fun, the true factored algorithm is:

$$\frac{\delta L}{\delta w_{0,i}^0} = \frac{2(y - \hat{y})(y - y^2)}{1 + e^{-(\sum_h (w_{i,h}^1 * y_h^2))}}$$

And now I'm tempted to factor it out down to the derivatives, but due to time, and space, I won't do that.

2.1.5 Derivative of Layer-1 Weights

Next, we need to find the derivative of each of the weights that connect the input layer to the hidden layer. We can write this derivative as follows:

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} * \frac{\delta w^0_{0,i}}{\delta y^1_i} * \frac{\delta y^1_i}{\delta w^1_{i,j}}$$

Thus, with using $\frac{\delta L}{\delta w^0_{0,i}}$, we can derive $\frac{\delta L}{\delta w^1_{i,j}}$. We will keep $\frac{\delta L}{\delta w^0_{0,i}}$ in as itself, since in our algorithms we will be able to store this value from the previous step.
For simplifcation, $\sigma(\ldots) = \sigma(W^1 * \vec{y^2}) = y^1_i$.

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} * (2(y-\hat{y})(y-y^2)(\sigma'(\ldots))) * (w^1_{i,j} * y^2_j)$$

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} * (2(y-\hat{y})(y-y^2)(\sigma(\ldots)(1-\sigma(\ldots)))) * (w^1_{i,j} * y^2_j)$$

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} * (2(y-\hat{y})(y-y^2)(y^1_i(1-y^1_i))) * (w^1_{i,j} * y^2_j)$$

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta w^0_{0,i}} * (2(y-\hat{y})(y-y^2)((y^1_i) - (y^1_i)^2) * (w^1_{i,j} * y^2_j)$$

3 Implementation

3.1 Data Importing

Since the data was given in a string csv format, an additional python script was created to import the data into two matricies, one contianing the inputs and one contianing the expected outputs.

4 Validation

5 Reflection