Data Science 210 Final Project

Thomas Kwashnak

Fall 2021

.

Contents

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

I've sectioned off the design of the algorithms into the independant algorithms, ordering them to try and explain the progression of deriving the algorithms.

2.1 Sigmoid Activation Function

In my 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.

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

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

When using this function on a matrix or vector, we simply just apply the sigma function for all individual values in the matrix/vector. The pseudocode algorithms of these functions are as follows:

Algorithm 1: $\sigma(n)$ function for both constants and matrices

```
Input: A, which can either be a constant, or a n x m matrix
Output: B, which is equivilant to σ(A)
if A is a Matrix then
    B =new Matrix(n x m) of 0s
    for i = 0, 1, 2...n do
        for j = 0, 1, 2...m do
            B[i][j] = σ(A[i, j])
        end
    end
    return B
else
    return 1/(1 + e^-A)
end
```

2.2 Forward Feeding Algorithm

The forward feeding algorithm is what we use in order to calculate the resulting number from a given input into the neural network. We are given an input layer: $\vec{y^2}$, and the following weight matrices W^0, W^1. Mathematically, the forward feeding algorithm is a recursive algorithm of the following structure:

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

Since we have two layers and an output layer, our function for the whole network is described as follows:

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

The complete algorithm, which can support any number of layers, is described below:

Algorithm 2: Forward-Feeding Algorithm

Input: Y_{in} - An h-length vector of the initial input values
Input: W - An array or list of N Weight-Matrices of varying sizes. The index of the matrix should relate to the order at which they are closest to the output layer. The size of W^a should be i x j, where i is the number of nodes in that layer, and j is the number of nodes for the layer W^{a+1}. The last Weight-Matrix should be of size i x h
Output: Y - The resulting vector of values outputted by the neural network
$Y =$ copy of Y_{in}
for $i = N-1, N-2...1, 0$ **do**
  $Y = \sigma(W^i \cdot Y)$
end
return Y

The ability for this algorithm to account for any number of inputs allows us to reuse it, regardless of whether we want the result from one of the hidden nodes or the whole network.

2.3 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.

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

2.4 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:

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

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

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

$$\frac{\delta L}{\delta w^0_{0,i}} = 2(y-\hat{y}) \cdot (\sigma(\ldots)(1-\sigma(\ldots))) \cdot (\sigma(\sum_h (w^1_{i,h} \cdot y^2_h))$$

$$\frac{\delta L}{\delta w^0_{0,i}} = 2(y-\hat{y}) \cdot (y \cdot (1-y)) \cdot (\sigma(\sum_h (w^1_{i,h} y^2_h))$$

$$\frac{\delta L}{\delta w^0_{0,i}} = 2(y-\hat{y}) \cdot (y-y^2) \cdot (\sigma(\sum_h (w^1_{i,h} \cdot y^2_h))$$

And because I couldn't resist it...

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

(I was about to factor it out even more...)

2.5 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:

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

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 $\frac{\delta L}{\delta w^0_{0,i}}$ as itself, since we will just be able to call it in our implementation.
For simplifcation, $\sigma(\ldots) = \sigma(W^1 \cdot \vec{y^2}) = y^1_i$.

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

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

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

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

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