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 $\sigma(A)$

```
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:

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

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 0^{th} Layer Weight Derivative

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.
Given the following variables:

$$y^1_{i,0} = \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0})$$

$$y^0_{0,0} = \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})$$

We can calculate the derivative as follows:

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

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

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

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

2.5 1^{st} Layer Weight Derivative

Next, we need to find the derivative of each of the weights that connect the input layer to the hidden layer. Given the following variables:

$$y^1_{i,0} = \sigma(\sum_h w^1_{i,h} \cdot y^2_{h,0})$$

$$y^0_{0,0} = \sigma(\sum_h w^0_{0,h} \cdot y^1_{h,0})$$

We can calculate the derivative as the following:

$$\frac{\delta L}{\delta w^1_{i,j}} = \frac{\delta L}{\delta y^0_{0,0}} \cdot \frac{\delta y^0_{0,0}}{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}} \cdot \frac{\delta \sum_h w^0_{0,h} \cdot y^1_{h,0}}{\delta y^1_{i,0}} \cdot \frac{\delta y^1_{i,0}}{\delta \sum_h w^1_{i,h} \cdot y^2_h} \cdot \frac{\delta \sum_h w^1_{i,h} \cdot y^2_{h,0}}{\delta w^1_{i,j}}$$

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

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

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

We can then reorder it as..

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

And substitute in the 0^{th} layer derivative!

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

2.6 Backwards Propagation

This is the heart of the whole lab project, backwards propagation. The goal of backwards propagation is to modify the values of the weights used in the neural network in order to "train" the network on a given output. Since we are trying to minimize the error term, we want to nudge the inputs in such a way that the output gets closer to the value we want. The algorithm below will iterate through a given. This algorithm is specifically designed for this given context, meaning it has been hard-coded for the 3 layer network that we have.

Algorithm 2: Backwards Propagation for a 2-layer neural network

Input: Y^2 - The input vector (matrix), dimensions m x 1.
W^1_{in} - The weight matrix, dimensions h x m, that represents the weights used as variables go from $Y^2 \to Y^0$
W^0_{in} - The weight matrix, dimensions 1 x h, that represents the weights used as variables go from $Y^1 \to Y^0$
S - The step coefficient to modify weights by, as a scalar value.
$\hat{y}$ - The expected result from Forward-Feeding

Output: W^1 - The modified weight matrix, describing the same thing as W^1_{in}
W^0 - The modified weight matrix, describing the same thing as W^0_{in}

$W^1 \leftarrow$ copy of W^1_{in}
$W^0 \leftarrow$ copy of W^0_{in}
$Y^1 \leftarrow \sigma(W^1 \cdot Y^2)$
$Y^0 \leftarrow \sigma(W^0 \cdot Y^1)$
for $i \leftarrow (0, 1, \ldots, h-1)$ **do**
  $\Delta W^0 \leftarrow S \cdot 2 \cdot (Y^0_{0,0} - \hat{y}) \cdot Y^0_{0,0} \cdot (1 - Y^0_{0,0}) \cdot Y^1_{i,0}$
  $W^0_{0,i} \leftarrow W^0_{0,i} + \Delta W^0$
  for $j \leftarrow (0, 1, \ldots, m-1)$ **do**
    $W^1_{i,j} \leftarrow W^1_{i,j} + \Delta W^0 \cdot W^0_{0,i} \cdot (1 - y^1_{i,0}) \cdot y^2_{j,0}$
  end
end
return W^1, W^0

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