Assignment 1

May 10, 2020

1 Forward and Backward Propagation

The purpose of a neural network model is to make good predictions, like any other machine learning model. In this we stack up various units called 'neurons' in multiple layers and each layer having one or many neurons. Each neuron acts like a miniature model with its own parameters that include weights and the bias term. The neural net, is a complex structure of these miniature models. We have three basic steps in training a neural network that include forward propagation, calculating loss and backward propagation. Lets analyse them further :

1.1 Forward Propagation

In this, lets say we have L hidden layers and thus in total we have L + 2 layers in total including input and output layers. Say we have an input X, then we take the input and feed it to our first layer. Each node of the input vector is connected to each activation unit in the first hidden layer. Then, we have the weights of each activation unit of the hidden layer equal in number to input vector X and we feed the weighted input to the hidden layer. Then, in the layer we add the bias term corresponding to each activation unit and apply the activation function. The final value or the output of this hidden layer 1 is fed to the next hidden layer and so on.
Thus, in this step we take the input, we pass the input to model and multiply with weights and add bias at every layer and find the calculated output of the model.

1.2 Calculating loss

Next, after we have propagated forward in the whole network, we find that our model outputs a value for the given input. This is called the predicted value. Now we want this predicted value to be as close to the output labelled in our train set.So we calculate the error between the predicted and the actual values of output and this error is called the loss. This is an approximation of how wrong our predictions are relative to the given output.

1.3 Back Propagation

After we have calculated loss, it's time to train our model. In this,the goal is to use optimization algorithms like gradient descent to update our parameters such that the loss is minimum. We know that gradient of our loss function is the vector that points in the direction of greatest steepness and therefore we want to repeatedly take steps in the opposite direction of the gradient to eventually arrive at the minimum. In back prop, we are moving the error backwards through the model. We move the error backwards via the same weights and biases as we did in the forward propagation. The goal is to calculate error attributed to each neuron and adjust its weights accordingly by shifting it back layer by layer. The larger the error attributed to a neuron, the more is the need to change the neuron's weights and biases.
Therefore,at each step, the goal is to calculate error in weights and bias and update the parameters using gradient descent(we are using this algorithm although there are many other). At each step of moving back, we calculate the partial derivative of the loss w.r.t. that parameters and then update the parameters at each iteration thus changing the weights and moving towards goal of minimum error.

2 Calculating Forward and Back Prop

2.0.1 Calculating forward and back prop for given diagram:

Say we have input layer with 4 elements $x = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \end{bmatrix}$ and hidden layer with activation units $a = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 \end{bmatrix}$ and the predicted output $\hat{y}$. Lets first calculate forward prop equations of first hidden layer, say weight $w_{i,j}$ connects the input unit i to the hidden layer unit j. Say for first hidden unit a_0, we will have :

$$z_0 = w_{0,0}x_0 + w_{1,0}x_1 + w_{2,0}x_2 + w_{3,0}x_3 + b_0$$

$$a_0 = \sigma(z_0)$$

$$z_1 = w_{0,1}x_0 + w_{1,1}x_1 + w_{2,1}x_2 + w_{3,1}x_3 + b_1$$

$$a_1 = \sigma(z_1)$$

$$z_2 = w_{0,2}x_0 + w_{1,2}x_1 + w_{2,2}x_2 + w_{3,2}x_3 + b_2$$

$$a_2 = \sigma(z_2)$$

$$z_3 = w_{0,3}x_0 + w_{1,3}x_1 + w_{2,3}x_2 + w_{3,3}x_3 + b_3$$

$$a_3 = \sigma(z_3)$$

$$z_4 = w_{0,4}x_0 + w_{1,4}x_1 + w_{2,4}x_2 + w_{3,4}x_3 + b_4$$

$$a_4 = \sigma(z_4)$$

Then we calculate the activation of the output layer or $\hat{y}$ as:

$$\hat{y} = \sigma(w_{0,0}^{[2]}a_0 + w_{1,0}^{[2]}a_1 + w_{2,0}^{[2]}a_2 + w_{3,0}^{[2]}a_3 + w_{4,0}^{[2]}a_4 + b_0^{[2]})$$

Then we calculate the loss of the output layer :

$$l = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

Then, calculating derivative of loss w.r.t to $\hat{y}$ and $z^{[2]}$:

$$da^{[2]} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$dz^{[2]} = \hat{y} - y$$

Propagating backwards, we have to calculate derivative w.r.t individuals parameters. From output layer to hidden layer, we have :

$$dw_{i,0}^{[2]} = dz^{[2]}.a_i for i = 0..4$$

$$db^{[2]} = dz^{[2]}$$

Then moving from this layer to backwards:

$$dz_i^{[1]} = w_{i,0}^{[2]} * dz^{[2]} * \sigma^{'}(z_i^{[1]}) for i = 0..4$$

$$dw_{i,j}^{[1]} = dz_j^{[1]}.x_i for i = 0..3, j = 0..4$$

$$db_i^{[1]} = dz_i^{[1]} for (i = 0..4)$$

Therefore, above are formulas calculating each units equations for forward and backward propagation. Then, we update these weights and run this process for the given number of training examples.

2.0.2 Vectorizing the above equations

We have $X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$ of dimension (4, 1) and hidden layer with $A^{[1]} = \begin{bmatrix} a_0^{[1]} \\ a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$ of dimension (5, 1) say we have bias $b^{[1]} = \begin{bmatrix} b_0^{[1]} \\ b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$ of dimension (5, 1) and weight matrix $W^{[1]} = \begin{bmatrix} w^{[1]}_{0,0} & .. & .. & w^{[1]}_{3,0} \\ : & & & \\ : & & & \\ : & & & \\ w^{[1]}_{0,4} & & & w^{[1]}_{3,4} \end{bmatrix}$ of dimension (5, 4). Then calculating forward propagation, we have :

$$Z^{[1]} = W^{[1]}.X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$

for the next layer, we have $A^{[1]}$ as input, then we have $Z^{[2]}$ as a (1, 1) dimension and $A^{[2]} = \hat{y}$ i.e. our predicted output. The matrix $W^{[2]} = \begin{bmatrix} w^{[2]}_{0,0} & .. & .. & .. & w^{[2]}_{4,0} \end{bmatrix}$ as a (1, 5) dimension vector. then we have:

$$Z^{[2]} = W^{[2]}.A^{[1]} + b^{[2]}$$
$$A^{[2]} = \hat{y} = \sigma(Z^{[2]})$$

We have our loss function same as above as we have one output unit, and thus propagating backwards we have:

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = dZ^{[2]} * A^{[1]}.T$$
$$db^{[2]} = dZ^{[2]}$$
$$dZ^{[1]} = (W^{[2]}.T * dZ^{[2]}) * \sigma^{'}(Z^{[1]})$$
$$dW^{[1]} = dZ^{[1]} * X.T$$
$$db^{[1]} = dZ^{[1]}$$

3 Activation functions and derivatives

Below are the activation functions and their derivatives respectively.

3.1 Sigmoid

3.1.1 Function:

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

3.1.2 Derivative:

$$\sigma^{'}(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$
$$\sigma^{'}(x) = \sigma(x) * (1 - \sigma(x))$$

3.2 Relu

3.2.1 Function:

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

i.e. $f(x) = maximum(0, x)$

3.2.2 Derivative:

$$f^{'}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

3.3 Leaky Relu

3.3.1 Function:

In this, we have a factor multiplying with x in case x is negative i.e. $f(x) = maximum(ax, x)$ and this a has to be ¡ 1 else it will pick the x when negative and ax when x is positive. Say a = 0.01, then :

$$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$$

3.3.2 Derivative:

$$f^{'}(x) = \begin{cases} 1 & x \geq 0 \\ 0.01 & x < 0 \end{cases}$$

3.4 Tanh

3.4.1 Function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3.4.2 Derivative:

$$\tanh^{'}(x) = 1 - (\frac{e^x - e^{-x}}{e^x + e^{-x}})^2$$

$$\sigma^{'}(x) = 1 - tanh(x)^2$$

3.5 Softmax

3.5.1 Function:

$$S_j = \frac{e^{a_j}}{\sum_{k=1}^{n} e^{a_k}} for j = 1....n$$

3.5.2 Derivative:

Since softmax is a function, the most general derivative we compute for it is the Jacobian matrix:

$$DS = \begin{bmatrix} D_1S_1 & .. & .. & D_1S_n \\ : & & & : \\ : & & & : \\ D_1S_n & .. & .. & D_nS_n \end{bmatrix}$$

Thus the derivative:

$$D_jS_i = \begin{cases} S_i(1 - S_j) & i = j \\ -S_iS_j & i \neq j \end{cases}$$