

1. Forward and back Propagation

Forward propagation in a neural network (NN) is set up similarly to in logistic regression. There is a matrix of weights for each layer of the network, which helps in traveling between the layers, as well as a bias vector for each layer. The input to the NN is a vector of input features, which is iterated over each training example. It is then multiplied (matrix multiplication) by the weights for that layer and the bias terms are added, and the whole is passed through an activation function. The output serves as the input for the next layer, and the process continues.

Back propagation is needed because unlike logistic regression, there are many layers and features in each layer, which makes it harder to pinpoint how much error is due to each layer and feature.

The general concept followed in back prop is that the total error in our output is found (our output - training set real output). This is the total error due to the last layer of our NN. Now this error is due to the error of the features in the previous layer, and so on. But how do we know what percentage of the error is due to each feature in the 2nd last layer? The concept followed is that the error of each feature is directly proportional to the contribution of that feature i.e. if a feature in a particular layer is contributing heavily to the output of that layer, then a major part of the error arising in that layer will also be because of that feature, and hence we need to adjust the weights for that feature accordingly.

3.

Sigmoid : $1/(1 + e^{(-x)})$

Derivative: $\text{sigmoid}(x) * (1 - \text{sigmoid}(x))$

ReLU: $y = x; x \geq 0$

$y = 0; x < 0$

Derivative: $1; x > 0$

$0; x < 0$

Not defined at 0

This also be defined as $\max(x, 0)$

And hence the derivative as $\max(0, x/\text{abs}(x))$

Leaky relu:

$\max(x, Ax)$, where $0 < A < 1$

Derivative: $\max(A, x/\text{abs}(x))$

Tanh:

$1 - 2/(e^{(2x)} + 1)$

Derivative:

$4/(e^x + e^{(-x)})^2$

Softmax:

For an array of features $X = [x(i)], 1 \leq i \leq n$

$$S(x(i)) = e^{x(i)} / \text{Sum}$$

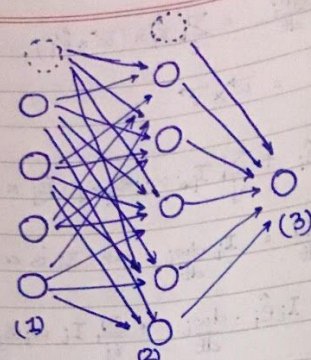
Where Sum = summation($e^{x(i)}$) over i

$$\text{Derivative } (d(S(x(i)))/d(x(j))) = S(x(i)) * \{ \delta_{ij} - S(x(i)) \}$$

Where δ_{ij} is the kronecker delta fn

i.e. 1 if $i = j$, 0 otherwise

2.



For this N

$O^{(1)}$ is a 5×5 mapping layer (1) to (2)

$O^{(2)}$ is a 1×6 matrix mapping layer (2) to (3)

- The dotted neurons are added to account for bias
- $g(x)$ is some general activation function, here sigmoid

$a^{(1)} = x^{(1)}$ // first layer is just inputs, (5×1) vector

$a^{(2)} = g(O^{(1)} \times a^{(1)})$ $a^{(2)}$ is (6×1) vector

$a^{(3)} = g(O^{(2)} \times a^{(2)})$ $a^{(3)}$ is (1×1) vector

$a^{(3)}$ is output, and we finished forward prop.

Now $S^{(3)} = a^{(3)} - y$ $\{1 \times 1 \text{ vector}\}$

$S^{(2)} = [O^{(2)}]^T \cdot S^{(3)}$ $\{6 \times 1 \text{ vector}\}$

there is no $S^{(1)}$ as there can be no error in our input layer as its input is the given training data

then the derivative of the cost function w.r.t the weight mapping the i^{th} feature of layer 'l' to j^{th} feature of 'l+1'

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = a_j^{(l)} \cdot \{ S_i^{(l+1)} \cdot (a_i^{(l+1)} \cdot (1 - a_i^{(l+1)})) \}$$

let us update $S_i^{(l+1)} = S_i^{(l+1)} \cdot (a_i^{(l+1)} \cdot (1 - a_i^{(l+1)}))$

then in vector form

$$\Delta^{(l)} = \Delta^{(l)} + \frac{1}{m} \cdot \{ S^{(l+1)} \cdot (a^{(l)})^T \}$$

where m is no. of training examples