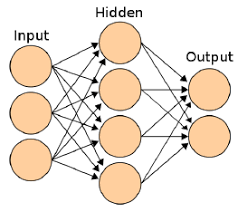
Name: Sarvesh Harikant

Email: [sarveshsh8@gmail.com](mailto:sarveshsh8@gmail.com)

Phone: 7022410051

MNIST DATASET USING 3 LAYER NEURAL NETWORK:



In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers

In this model we are using multilayer perceptron model which consists of 3 layers i.e

1. Input Layer
2. Hidden Layer
3. Output Layer

Input layer -

In this model the size of the input layer is 28\*28 = 784. The input data is passed through each neuron and then I have used relu as an activation function which converts value between 0 to x. Thus the output is then passed into the hidden layer.

Y1 = w1\*x1 + w2\*x2 + W3\*X3 + b1 + b2 + b3 (for a single node)

(Suppose if the weights value get initialised to 0, the biases are added to atleast give some value instead of becoming a dead neuron.)

Relu = max (0,x) i.e 0 when x<0 and x when x>0

Ex: when x value is 3, then it is initialised as max (0,3)

Hidden layer –

Hidden layer is present is the middle i.e second layer. The outputs which we got from the input layer is taken as an input in the hidden layer. It is then multiplied with weights and biases and then also I have added a drop out layer to avoid overfitting of the data.

In this layer I have used softmax activation function to get the desirable outputs. Since I have 10 different outputs, sigmoid or any other activation cannot be used since it has multiple output. That’s the reason I have used softmax which give you values in vector when added together we get a value as 1. But the desirable output has a highest value i.e more than 0.5

Output Layer:

So since we have 10 different targets, it gives us 10 different values.

**How the weights get initialized:**

The weights are initialized according to the activation function we use in the layers.

So in the first layer we have used relu as an activation function, the weight initialization technique generally used for relu is he-init (normal or uniform), but in my python code I have used it as:

First layer I will get values in terms of [784x128] which I will then divide each value by sqrt(784+128) = 30.1 which follows standard normal distribution (SND). Here np.random.randn converts the values to follow SND. Biases are also calculated by following method but it will be in terms of ([1x128].

**Activation Function:**

I have also used multiple activation function in my python code such as (tanh, sigmoid, relu).

Tanh function which converts value between 1- and 1. Derivative value is 1.

Sigmoid which converts between 0 to 1. (Derivative value between 0 and 0.25)

Softmax Af is used when we have multiple outputs.

**Loss Function:**

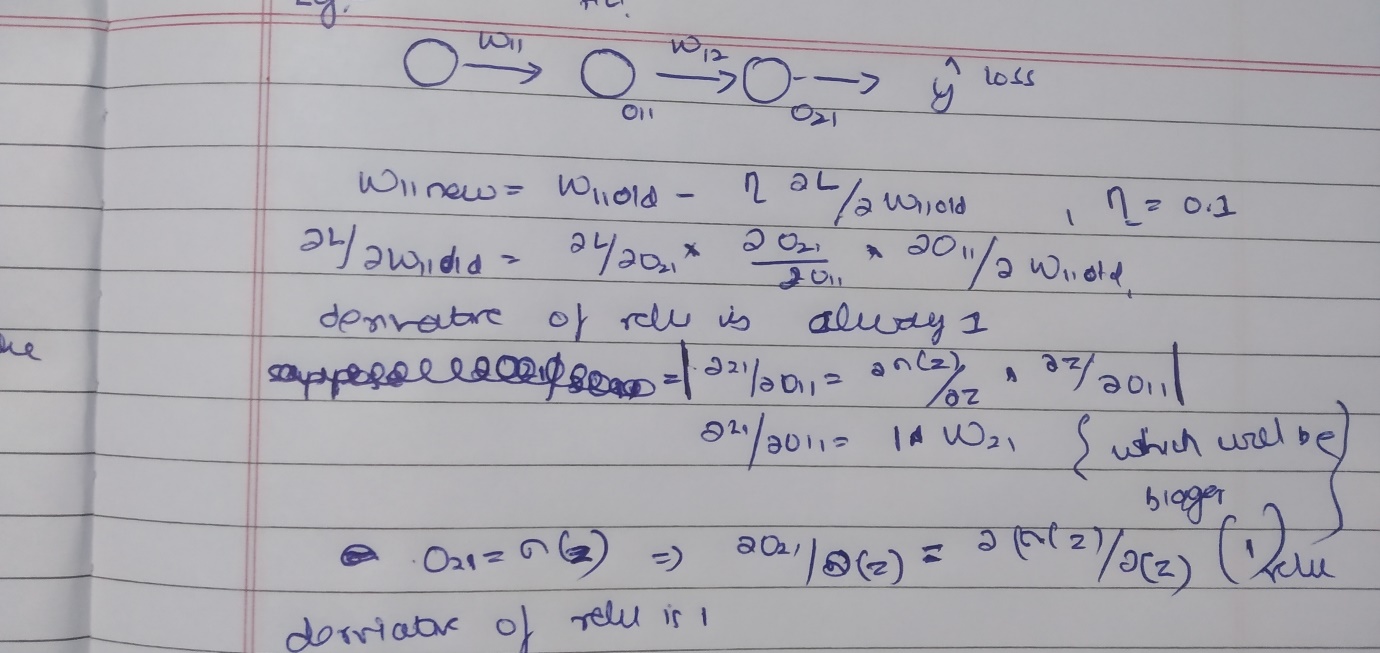
The loss function which I have used is categorical cross entropy. The loss function categorical cross entropy is used to quantify model error in single and multi-class classification problems.

**Backward Propagation:**

Backward propagation is used to update the weights and reduce loss function.

In my model I have used relu as an activation function which doesn’t have a vanishing gradient problem. The derivative of relu is 1 which makes it better than other activation function in updating weights and reducing loss. In most cases relu is used when there is large neural network, whereas in my 3layer neural network model we could used sigmoid activation function also. In my python code based model, I have used all 3 activation function, in which comparatively relu gave me a better accuracy.

In my diagram, I have just take one network to explain how backpropagation works. To update the weights with a learning rate of 0.1 and as we know the derivative of relu is 1, it helped me in reducing the loss function as well.

****

**Classification Matrix:**

precision recall f1-score support

0 0.99 0.99 0.99 980

1 0.99 0.99 0.99 1135

2 0.98 0.98 0.98 1032

3 0.98 0.99 0.98 1010

4 0.98 0.99 0.98 982

5 0.98 0.98 0.98 892

6 0.99 0.98 0.99 958

7 0.98 0.98 0.98 1028

8 0.99 0.97 0.98 974

9 0.98 0.98 0.98 1009

accuracy 0.98 10000

macro avg 0.98 0.98 0.98 10000

weighted avg 0.98 0.98 0.98 10000

From my vanilla cnn model, I have got a f1 score as 0.99 which is good. That mans my model has performed well with an accuracy of 98%.

