

1. DESCRIPTION OF THE ipynb NOTEBOOK:

Solution: Details about the classes and function used in Assignment-1:

1.plot_confusion_mtx(matrix , num_classes)

Function takes argument as the object returned from `sklearn.metrics.confusion_matrix()` as ‘matrix’ and total classes (here 10) as ‘num_classes’, and plots the confusion matrix.

2.plot_images()

Function plots the unique classes (0-9) images from the Fashion_Mnist dataset.

3.class Activation

- General class for activation functions that are going to be used in Neural Network. The functions defined inside the class are `sigmoid(x)` , `Softmax(x)` ,`Relu(x)` ,`Tanh(x)` ,`Derivative_Sigmoid(x)` ,`Derivative_Relu(x)` ,`Derivative_Tanh(x)`.
- For `Softmax()` the stable version is used which will avoid overflow.

4.class Loss

- **CategoricalCrossEntropy(ypred,y)** Takes the probability values as `ypred` and true values as one hot encoded vectors and returns the average `CategoricalCrossEntropyLoss` over all datapoints.
- **MSE(ypred,y)** Arguments are same as above but it returns the average squared loss taken over all the datapoints.

5.class Layer

5.a.Layer class will be having the following attributes related to a Layer:

- `num_inputs` : The number of inputs / the number of neurons in previous layer.
- `num_neurons` : The number of neurons in the layer.
- `activation` : The postactivation function used for all neurons in Layer.
- `optimizer` : The optimizer used.
- `loss` : The loss (Categorical crossentropy or Squared Error).

- a_L : The preactivation values.
- h_L : The postactivation values.
- W : The weight matrix provided to the layer.
- b : the biases provided to the layer.
- del_h : The gradient of loss w.r.t the post activations of given layer.
- del_a : The gradient of loss w.r.t the pre activations of given layer.
- $\text{del}W$: The gradient of loss w.r.t the weights of given layer.
- $\text{del}b$: The gradient of loss w.r.t the biases of given layer.
- prev_Uw , prev_Ub , prev_Vw , prev_Vb , prev_Mw , prev_Mb : Stores the previous updates which are going to be used in different optimizers.
- $\text{layerInitialization}$: Used to initialize the weights and biases of given layer.

5.b Layer class is having the following Function

- **Forward(h_{L-1})** Given the input to the Layer it computes the preactivation (a_L) and post activation (h_L) for that layer.
- **get_gradients(isoplayer,Y_hat,Y,prev_hL,nextW,next_del_a)**
If isoplayer=1 it calculates the gradients($\Delta_h(L)$, $\Delta_a(L)$, $\Delta_W(L)$, $\Delta_b(L)$) for the output layer else for hidden layers.

6.class IpLayer

creates the input layer.

7.class NN

- The attributes for this class are Hiddenlayeractivation ,outputlayeractivation, input data shape , output shape(number of classes),optimizer used, ,epochs ,batch_size, η , ϵ , β , β_1 , γ ,weightDecay etc.
- It has a $\text{Layer}=\emptyset$ list which contains the Layers related to the NeuralNetwork.initialization of layer happens in the constructor of NN class.
- **Feed_Forward(X)** Given a Batch of data X this function forward propagates the data and calculates Y_{hat} (batchsize x 10).(the probability values).

- **Back_Propogation(Y)** Given Y and Y_hat it computes the gradients for each layer in Layers=[] using get_gradients() method for each layer.
- **Update_Params(itearion)** For each update(iteration) in a given epoch it updates the W and b for each layer according to the optimizer implemented.
- **Final_Prediction(X)** Takes the batch and calculate the output probabilities with the current values of W and b for the Neural network.

8.MINI_BATCH(NN,X,Y)

It takes the object of NN class and perform the minibatch depending upon the size of the batch(stored in NN.batch_size).For doing Stochastic initialize the NN.batch_size=1 and for performing Batch gradient descent initialize NN.batch_size=N(number of data-points).

In 1 pass of batch it Do the following:

- Feed Forward the batch and calculate Y_hat
- Back propogate to get the gradients for each layer
- Update the W and b for each layer

It returns the predicted probabilities of the X after running for epochs.

Instructions on How to train the model with certain parameters:

To implement any model of your choice the params can be changed (the different values for each parameter as placed next to it in the ipynb file).Then call the Run_Model() function with train ,cv and test data.

```
params={'input_shape':784,
        'num_hlayers':3,
        'neurons':[32,32,32],
        'HLAc':'ReLU',
        'OPAc':'Softmax',
        'loss':'squareerror',
        'optimizer':'momentum',
        'eta':1e-3,
        'batch_size':64,
        'epochs' :2,
        'layer_init':'Xavier',
        'weightDecay':0.0005,
        'output_shape':10}
```

```
Run_Model(params,X_actualTrain,Y_actualTrain,X_validationData,  
Y_validationData,X_test,Y_test)
```