COMP5329 - Deep Learning - Assignment 1
Team : Andrew Zhang: (SID 500543568), Vincent Yunansan (SID 530454874)

A. BACKGROUND:
Goal of this assignment is to implement a neural network without the use of modern machine learning libraries. Teams are provided with training and test data, each with 50,000 and 10,000 data points across 10 classes. Each data point consists of 128 floating numbers.

B. CODE OVERVIEW:
The codes can be ran top to bottom. The code was originally run on a local CPU due to the bandwidth limitation on Colab. All relevant functions and helper functions are placed at the top, followed by the best model recommendation and hyperparameter testing at the bottom.

The structure of our neural network implementation can be divided into three main parts, explained below:

B1. PRE-PROCESSING:

1. Normalization: training and testing data are normalised (mean = c.0, standard deviation = c.1).

2. resize_and_shuffle: (i) shuffle: the dataset provided was sorted by class, shuffling the data is important especially if we are using mini-batch. (ii) resize: we implemented the capability to downsize the dataset, so we do not have to run our training and testing with the full dataset during the building phase.

SAMPLE CODE START

```
# Explanation: this command resizes the training data by 0.5 x 50,000 = 25,000 with a set seed for replicability.

data, label = resize_and_shuffle(input_data, input_label, proportion = 0.5, random_state = 42)
```

##############################

3. train_val_split: splits the data into training and validation sets, especially important if we were to implement early stopping.

SAMPLE CODE START

```
# Explanation: this command splits the training data to 0.8 x 25,000 = 20,000 in the training data and 0.2 x 25,000 = 5,000 in the validation data with a set seed for replicability.
```

```
train_data, train_label, val_data, val_label =
train_val_split(input, output, proportion = 0.8, random_state = 42)

############################
```

B2. THE DEEP LEARNING MODEL:
The deep learning model and its methods are implemented in three classes:

1. Activation class: this object contains our activation functions and their derivatives. This class is called by HiddenLayer when the neural network is constructed. This class contains:
- Activations for hidden layers include tanh, sigmoid, relu, leaky relu
- Activation for output layer is softmax.

2. HiddenLayer class: this object contains all parameters and methods relevant to the hidden layer. In summary, this class contains:
- Weights, biases, and their gradients
- forward and backward passes
- Implementation of momentum, weight decay, dropout, batch normalisation, and Adam.

3. MLP class: this object contains the main constructor and iterator for our neural network. In summary, this class contains:
- initial neural network constructor
- forward and backward: which will trigger the forward and backward propagations in all layers
- criterion cross entropy: which is our objective function
- update: which will update weights and gradients after forward and backward propagation
- getBatch: which functions as a data loader for mini batch
- fit: which will fit our model with the training data, do periodical validation checks, trigger early stopping (if used),
- predict: which predicts labels based on the trained neural network
- eval: which evaluates model performance during training
- train_loop: which implements a single forward, backward, and update loop, to be used in fit.

```
##### SAMPLE CODE START #####

# Explanation: This constructs a neural network object with two
hidden layers (relu, 64 nodes each) and one output layer (softmax,
10 nodes).

nn = MLP([128,64,64,10], [None, 'relu','relu','softmax'])

############################

##### SAMPLE CODE START #####

# Explanation : this implements a fit with training and validation
```

data and labels, with different parameters (e.g. learning_rate, epochs, batch_size) and methods. Further explanation is provided in comments within the code file

```
train_loss, val_loss, epoch = nn.fit(train_data, train_label, val_data, val_label, momentum_gamma = 0.0, learning_rate = 0.001, epochs = 200, batch_size = 100, weight_decay = 0.0, dropout_rate = 0.0, early_stopping = [3,10], batchnorm_switch = False, adam_switch = False, adam_learning_rate = 0.0)
```

```
##############################
```

B3. HELPER FUNCTIONS:
1. Hyperparameter_testing [IMPORTANT]: a hyperparameter tuning function is implemented to simplify the repeated hyperparameter testing. This function takes all possible combination of parameters as lists and iterates through them. This function also calculates training and testing scores, records the number of epochs for each training run, and runtimes

```
##### SAMPLE CODE START #####

# Explanation: n the above example, hyperparameter tuning was performed over early_stopping combination, e.g. ([np.inf,10] for no early stopping, [3,10] for early stopping with 3 max cycle of deteriorating validation score and validation checks every 10 epochs.this function returns relevant parameters and scores for reporting purposes and train and validation losses for plotting purposes. Further explanation is provided in comments within the code file

learning_rates = [0.001]
batch_sizes = [1]
early_stopping_combination = [[np.inf,10],[3,10]]
epoch_counts = [200]
momentum_gammas = [0.0]
weight_decays = [0.0]
dropouts = [0.0]
batchnorm_switches = [False]
adam_learning_rates = [0.00]
node_counts = [[128,64,64,10]]
node_activations = [[None,'relu','relu','softmax']]

parameters = (learning_rates,batch_sizes,early_stopping_combination, epoch_counts, momentum_gammas, weight_decays, dropouts, batchnorm_switches,adam_learning_rates, node_counts,node_activations)

results = hyperparamater_testing(parameters,train_data, train_label, val_data, val_label)

##############################
```

2. filter_asset: this function creates a data frame from

hyperparameter tuning results, for reporting purposes. users can choose how to filter the dataset (e.g. only include results where learning_rate = 0.001) and which columns to show in the data frame.

3. multi_plotter: This function plots training loss curves with customisable color, marker, and line. list down all column names in label_keys to display them in the legend box. add the name of the label in marker_param, line_param, and color_param to add them as customisation.

SAMPLE CODE START

```
target ='train_loss'
label_keys =['early_stopping']
marker_param = None
line_param = 'early_stopping'
color_param = 'early_stopping'
title = 'Training Losses with and without early stopping'
multi_plotter(title, assets_exp_1, target, label_keys, marker_param,
line_param, color_param, fig_size =(10,3))
```

##############################