Training on your own Model and on your own Data

Example:

In order to use the code for creating your own model on your own dataset, you would have to edit the code under the following headings in 'NN_Training_Code.ipynb:

- 'Class *Dataset*' : create an object of class 'Dataset' and call the method by setting each argument train, valid and test separately on the object to get the training, validation and testing data.

```
#path to dataset
dir_world ='/content/World_Data_Transformed.csv'

#remove these features from the dataset
dataset_world = Dataset(dir=dir_world,
remove_cols=['location', 'dates'])

#Scale dataset
dataset_world.scale_dataset(st=True)

#Training Data
train_data_world = dataset_world.split_data(train=True)

#Validation Data
valid_data_world = dataset_world.split_data(valid=True)

#Testing Data
test_data_world = dataset_world.split_data(test=True)
```

- '*Model*': create an object of class 'Network' by passing a list of numbers where each index will represent a layer and value will represent number of neurons, activation function training and validation data. Furthermore, call the method 'train' on the object to start the training process

```
hidden_units_world = [6,4,3,2]
activation_function_world = 'Sigmoid'
```

```
model_world =
Network(hidden_units_world,activation_function_world,
train_data_world, valid_data_world)

#setting up hyperparameters
batch_size_world = 50
epochs_world = 10
lr_world = 0.01
lmbda_world = 0.2

train_cost_iter_world, valid_cost_iter_world =
model_world.train(batch_size_world, epochs_world, lr_world,
lmbda_world )
```

Layers:

A detailed information on which layers are supported by the code and what they expect is provided below:

Dataset

```
class Dataset:
  #constructor
 def __init__(self,dir,remove_cols=[]):

   # args:
   # dir = (str)directory to dataset with last column as target values
   # remove_cols = (list) name of columns which are not to be added in the
dataset as list

def scale_dataset(self,st):

   #this function carries out scaling of the features
   #it can either be between 0 and 1 or between -1 and 1
   #set st=True for [0,1] and st=False for [-1,1]


def split_data(self,train=False, test=False, valid=False):
```

```
   # this fucntion carries out the splitting of data into train, test, and
validation datasets
   # depending upon which arg is set True.
   # Splitting ratio: 60% for  train, 20% for test, 20% for validation
   # Only one arg should be set true when calling this function
   # otherwise the dataset of the arg which is in first in arg list will
be returned.
```

Network

```
class Network:

 #constructor
 def __init__(self, hidden_units, activation_fnt, train_data, valid_data):

   #args:
   #     hidden units: (list) it include both number of hidden layers and
number of units in each hidden layer as list
   # e.g. for a network with two hidden layers, first hidden layer has 3
units and second hidden layer has 2 units
   # hidden_units=[3, 2]
   #
   #     activation_fnt: (str) which will be applied to the hidden layers
   # its value can either be 'Sigmoid' or 'ReLU'
   #
   #     train_data: (tuple) containing both X and Y as tuple e.g.
(X_train, Y_train)
   #     valid_data: (tuple) containing both X and Y as tuple e.g.
(X_valid, Y_valid)


#train the neural network
 def train(self, batch_size, epochs, lr, lmbda):

   #args:
   #      batch_size = (int) no of examples after which weights will be
updated
```

```
    #       epochs = (int) no of times complete dataset needs to be passed
through the neural network during training
    #       lr = (float/int) learning rate
    #       lmbda = (float/int) regularization parameter
```