# Group B Deep Learning Assignment No: 2A

**Title of the Assignment:** Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition

**Objective of the Assignment:** Students should be able to solve Multiclass classification using Deep Neural Networks.

Prerequisite:

1. Basic of programming language
2. Concept of Multi Classification
3. Concept of Deep Neural Network

## ---------------------------------------------------------------------------------------------------------------

**Contents for Theory:**

1. What is multi-Classification
2. Example of Multi-Classification
3. How Deep Neural Network Work on Multi-Classification
4. Code Explanation with Output

## ---------------------------------------------------------------------------------------------------------------

**What is multiclass classification?**

Multi Classification, also known as multiclass classification or multiclass classification problem, is a type of classification problem where the goal is to assign input data to one of three or more classes or categories. In other words, instead of binary classification, where the goal is to assign input data to one of two classes (e.g., positive or negative), multiclass classification involves assigning input data to one of several possible classes or categories (e.g., animal species, types of products, etc.).

In multiclass classification, each input sample is associated with a single class label, and the goal of the model is to learn a function that can accurately predict the correct class label for new, unseen input data. Multiclass classification can be approached using a variety of machine learning algorithms, including decision trees, support vector machines, and deep neural networks.

Some examples of multiclass classification problems include image classification, where the goal is to classify images into one of several categories (e.g., animals, vehicles, buildings), and text classification, where the goal is to classify text documents into one of several categories (e.g., news topics, sentiment analysis).

**Example of multiclass classification-**

Here are a few examples of multiclass classification problems:

**Image classification:** The goal is to classify images into one of several categories. For example, an image classification model might be trained to classify images of animals into categories such as cats, dogs, and birds.

**Text classification:** The goal is to classify text documents into one of several categories. For example, a text classification model might be trained to classify news articles into categories such as politics, sports, and entertainment.

**Disease diagnosis:** The goal is to diagnose patients with one of several diseases based on their symptoms and medical history. For example, a disease diagnosis model might be trained to classify patients into categories such as diabetes, cancer, and heart disease.

**Speech recognition:** The goal is to transcribe spoken words into text. A speech recognition model might be trained to recognize spoken words in several languages or dialects.

**Credit risk analysis:** The goal is to classify loan applicants into categories such as low risk, medium risk, and high risk. A credit risk analysis model might be trained to classify loan applicants based on their credit score, income, and other factors.

In all of these examples, the goal is to assign input data to one of several possible classes or categories.

Multiclass classification is a common task in machine learning and can be approached using a variety of algorithms, including decision trees, support vector machines, and deep neural networks.

**Source Code and Output**

import numpy as np from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Dropout from tensorflow.keras.optimizers import RMSprop from tensorflow.keras.datasets import mnist import matplotlib.pyplot as plt from sklearn import metrics

# Load the OCR dataset

# The MNIST dataset is a built-in dataset provided by Keras.

# It consists of 70,000 28x28 grayscale images, each of which displays a single handwritten digit from 0 to 9.

# The training set consists of 60,000 images, while the test set has 10,000 images.

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# X\_train and X\_test are our array of images while y\_train and y\_test are our array of labels for each image.

# The first tuple contains the training set features (X\_train) and the training set labels (y\_train).

# The second tuple contains the testing set features (X\_test) and the testing set labels (y\_test).

# For example, if the image shows a handwritten 7, then the label will be the intger 7.

plt.imshow(x\_train[0], cmap='gray') # imshow() function which simply displays an image.

plt.show() # cmap is responsible for mapping a specific colormap to the values found in the array that you passed as the first argument.

# This is because of the format that all the images in the dataset have:

# 1. All the images are grayscale, meaning they only contain black, white and grey.

# 2. The images are 28 pixels by 25 pixels in size (28x28).

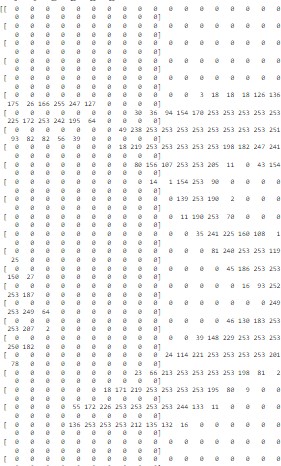
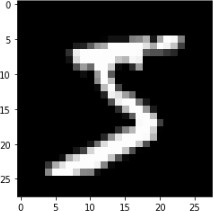
print(x\_train[0])

# image data is just an array of digits. You can almost make out a 5 from the pattern of the digits in the array.

# Array of 28 values

# a grayscale pixel is stored as a digit between 0 and 255 where 0 is black, 255 is white and values in between are different shades of gray.

# Therefore, each value in the [28][28] array tells the computer which color to put in that position when.



# reformat our X\_train array and our X\_test array because they do not have the correct shape.

# Reshape the data to fit the model print("X\_train shape", x\_train.shape) print("y\_train shape", y\_train.shape) print("X\_test shape", x\_test.shape) print("y\_test shape", y\_test.shape)

# Here you can see that for the training sets we have 60,000 elements and the testing sets have 10,000 elements.

# y\_train and y\_test only have 1 dimensional shapes because they are just the labels of each element. # x\_train and x\_test have 3 dimensional shapes because they have a width and height (28x28 pixels) for each element.

# (60000, 28, 28) 1st parameter in the tuple shows us how much image we have 2nd and 3rd parameters are the pixel values from x to y (28x28)

# The pixel value varies between 0 to 255.

# (60000,) Training labels with integers from 0-9 with dtype of uint8. It has the shape (60000,).

# (10000, 28, 28) Testing data that consists of grayscale images. It has the shape (10000, 28, 28) and the dtype of uint8. The pixel value varies between 0 to 255.

# (10000,) Testing labels that consist of integers from 0-9 with dtype uint8. It has the shape (10000,). X\_train shape (60000, 28, 28) y\_train shape (60000,) X\_test shape (10000, 28, 28) y\_test shape (10000,)

# X: Training data of shape (n\_samples, n\_features)

# y: Training label values of shape (n\_samples, n\_labels)

# 2D array of height and width, 28 pixels by 28 pixels will just become 784 pixels (28 squared). # Remember that X\_train has 60,000 elemenets, each with 784 total pixels so will become shape (60000, 784).

# Whereas X\_test has 10,000 elements, each with each with 784 total pixels so will become shape (10000, 784).

x\_train = x\_train.reshape(60000, 784) x\_test = x\_test.reshape(10000, 784) x\_train = x\_train.astype('float32') # use 32-bit precision when training a neural network, so at one point the training data will have to be converted to 32 bit floats. Since the dataset fits easily in RAM, we might as well convert to float immediately.

x\_test = x\_test.astype('float32') x\_train /= 255 # Each image has Intensity from 0 to 255 x\_test /= 255

# Regarding the division by 255, this is the maximum value of a byte (the input feature's type before the conversion to float32),

# so this will ensure that the input features are scaled between 0.0 and 1.0.

# Convert class vectors to binary class matrices num\_classes = 10

y\_train = np.eye(num\_classes)[y\_train] # Return a 2-D array with ones on the diagonal and zeros elsewhere. y\_test = np.eye(num\_classes)[y\_test] # f your particular categories is present then it mark as 1 else 0 in remain row

# Define the model architecture model = Sequential() model.add(Dense(512, activation='relu', input\_shape=(784,))) # Input cosist of 784 Neuron ie 784 input,

512 in the hidden layer model.add(Dropout(0.2)) # DROP OUT RATIO 20% model.add(Dense(512, activation='relu')) #returns a sequence of another vectors of dimension 512 model.add(Dropout(0.2)) model.add(Dense(num\_classes, activation='softmax')) # 10 neurons ie output node in the output layer.

# Compile the model model.compile(loss='categorical\_crossentropy', # for a multi-class classification problem

optimizer=RMSprop(), metrics=['accuracy'])

# Train the model batch\_size = 128 # batch\_size argument is passed to the layer to define a batch size for the inputs.

epochs = 20 history = model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=epochs, verbose=1, # verbose=1 will show you an animated progress bar eg. [==========] validation\_data=(x\_test, y\_test)) # Using validation\_data means you are providing the

training set and validation set yourself, # 60000image/128=469 batch each

# Evaluate the model score = model.evaluate(x\_test, y\_test, verbose=0) print('Test loss:', score[0]) print('Test accuracy:', score[1])

Test loss: 0.08541901409626007

Test accuracy: 0.9851999878883362

**Conclusion**- In this way we can do Multi classification using DNN.

## Assignment Question

1. What is Batch Size?
2. What is Dropout?
3. What is RMSprop?
4. What is the SoftMax Function?
5. What is the Relu Function?

**Conclusion**- In this way we can classify the Movie Reviews by using DNN.

**Group B Deep Learning**

**Assignment No: 2B**

**Title of the Assignment:** Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews. Use IMDB dataset

**Objective of the Assignment:** Students should be able to Classify movie reviews into positive reviews and "negative reviews on IMDB Dataset.

**Prerequisite:**

1. Basic of programming language
2. Concept of Classification
3. Concept of Deep Neural Network

# ---------------------------------------------------------------------------------------------------------------

**Contents for Theory:**

1. What is Classification
2. Example of Classification
3. How Deep Neural Network Work on Classification
4. Code Explanation with Output

**What is Classification?**

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, or to classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

# How Deep Neural Network Work on Classification-

Deep neural networks are commonly used for classification tasks because they can automatically learn to extract relevant features from raw input data and map them to the correct output class.

The basic architecture of a deep neural network for classification consists of three main parts: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, which is usually preprocessed to a fixed size and format. The hidden layers are composed of neurons that apply linear transformations and nonlinear activations to the input features to extract relevant patterns and representations. Finally, the output layer produces the predicted class labels, usually as a probability distribution over the possible classes.

During training, the deep neural network learns to adjust its weights and biases in each layer to minimize the difference between the predicted output and the true labels. This is typically done by optimizing a loss function that measures the discrepancy between the predicted and true labels, using techniques such as gradient descent or stochastic gradient descent.

One of the key advantages of deep neural networks for classification is their ability to learn hierarchical representations of the input data. In a deep neural network with multiple hidden layers, each layer learns to capture more complex and abstract features than the previous layer, by building on the representations learned by the earlier layers. This hierarchical structure allows deep neural networks to learn highly discriminative features that can separate different classes of input data, even when the data is highly complex or noisy.

Overall, the effectiveness of deep neural networks for classification depends on the choice of architecture, hyperparameters, and training procedure, as well as the quality and quantity of the training data. When trained properly, deep neural networks can achieve state-of-the-art performance on a wide range of classification tasks, from image recognition to natural language processing.

**IMDB Dataset-**The IMDB dataset is a large collection of movie reviews collected from the IMDB website, which is a popular source of user-generated movie ratings and reviews. The dataset consists of 50,000 movie reviews, split into 25,000 reviews for training and 25,000 reviews for testing.

Each review is represented as a sequence of words, where each word is represented by an integer index based on its frequency in the dataset. The labels for each review are binary, with 0 indicating a negative review and 1 indicating a positive review.

The IMDB dataset is commonly used as a benchmark for sentiment analysis and text classification tasks, where the goal is to classify the movie reviews as either positive or negative based on their text content. The dataset is challenging because the reviews are often highly subjective and can contain complex language and nuances of meaning, making it difficult for traditional machine learning approaches to accurately classify them.

Deep learning approaches, such as deep neural networks, have achieved state-of-the-art performance on the IMDB dataset by automatically learning to extract relevant features from the raw text data and map them to the correct output class. The IMDB dataset is widely used in research and education for natural language processing and machine learning, as it provides a rich source of labeled text data for training and testing deep learning models.

# Source Code and Output-

# The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users that are labeled as either positive (1) or negative (0).

# The reviews are preprocessed and each one is encoded as a sequence of word indexes in the form of integers.

# The words within the reviews are indexed by their overall frequency within the dataset. For example, the integer “2” encodes the second most frequent word in the data.

# The 50,000 reviews are split into 25,000 for training and 25,000 for testing.

# Text Process word by word at diffrent timestamp ( You may use RNN LSTM GRU )

# convert input text to vector reprent input text

# DOMAIN: Digital content and entertainment industry

# CONTEXT: The objective of this project is to build a text classification model that analyses the customer's sentiments based on their reviews in the IMDB database. The model uses a complex deep learning model to build an embedding layer followed by a classification algorithm to analyse the sentiment of the customers.

# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by sentiment

(positive/negative).

# Reviews have been preprocessed, and each review is encoded as a sequence of word indexes

(integers).

# For convenience, the words are indexed by their frequency in the dataset, meaning the for that has index 1 is the most frequent word.

# Use the first 20 words from each review to speed up training, using a max vocabulary size of 10,000. # As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input text parameters to determine the customer sentiments.

import numpy as np import pandas as pd from sklearn.model\_selection import train\_test\_split #loading imdb data with most frequent 10000 words from keras.datasets import imdb

(X\_train, y\_train), (X\_test, y\_test) = imdb.load\_data(num\_words=10000) # you may take top 10,000 word frequently used review of movies other are discarded

#consolidating data for EDA Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics data = np.concatenate((X\_train, X\_test), axis=0) # axis 0 is first running vertically downwards across rows (axis 0), axis 1 is second running horizontally across columns (axis 1), label = np.concatenate((y\_train, y\_test), axis=0)

X\_train.shape

(25000,)

X\_test.shape (25000,) y\_train.shape (25000,) y\_test.shape (25000,) print("Review is ",X\_train[0]) # series of no converted word to vocabulory associated with index print("Review is ",y\_train[0])

Review is [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14,

394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114,

9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5,

89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255,

5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64,

1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95] Review is 0 vocab=imdb.get\_word\_index() # Retrieve the word index file mapping words to indices print(vocab)

{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders':

16115, y\_train array([1, 0, 0, ..., 0, 1, 0]) y\_test array([0, 1, 1, ..., 0, 0, 0])

# Function to perform relevant sequence adding on the data

# Now it is time to prepare our data. We will vectorize every review and fill it with zeros so that it contains exactly 10000 numbers.

# That means we fill every review that is shorter than 500 with zeros.

# We do this because the biggest review is nearly that long and every input for our neural network needs to have the same size.

# We also transform the targets into floats.

# sequences is name of method the review less than 10000 we perform padding overthere # binary vectorization code:

# VECTORIZE as one cannot feed integers into a NN

# Encoding the integer sequences into a binary matrix - one hot encoder basically

# From integers representing words, at various lengths - to a normalized one hot encoded tensor (matrix) of 10k columns def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it with zeros so that it contains exactly 10,000 numbers.

# Create an all-zero matrix of shape (len(sequences), dimension) results = np.zeros((len(sequences), dimension)) for i, sequence in enumerate(sequences):

results[i, sequence] = 1

return results

# Now we split our data into a training and a testing set.

# The training set will contain reviews and the testing set

# # Set a VALIDATION set

test\_x = data[:10000] test\_y = label[:10000] train\_x = data[10000:] train\_y = label[10000:] test\_x.shape (10000,) test\_y.shape (10000,) train\_x.shape (40000,) train\_y.shape (40000,) print("Categories:", np.unique(label)) print("Number of unique words:", len(np.unique(np.hstack(data)))) # The hstack() function is used to stack arrays in sequence horizontally (column wise).

Categories: [0 1]

Number of unique words: 9998 length = [len(i) for i in data] print("Average Review length:", np.mean(length)) print("Standard Deviation:", round(np.std(length)))

# The whole dataset contains 9998 unique words and the average review length is 234 words, with a standard deviation of 173 words.

Average Review length: 234.75892

Standard Deviation: 173

# If you look at the data you will realize it has been already pre-processed.

# All words have been mapped to integers and the integers represent the words sorted by their frequency.

# This is very common in text analysis to represent a dataset like this. # So 4 represents the 4th most used word, # 5 the 5th most used word and so on...

# The integer 1 is reserved for the start marker, # the integer 2 for an unknown word and 0 for padding.

# Let's look at a single training example:

print("Label:", label[0]) Label: 1

print("Label:", label[1]) Label: 0 print(data[0])

# Retrieves a dict mapping words to their index in the IMDB dataset. index = imdb.get\_word\_index() # word to index

# Create inverted index from a dictionary with document ids as keys and a list of terms as values for each document

reverse\_index = dict([(value, key) for (key, value) in index.items()]) # id to word

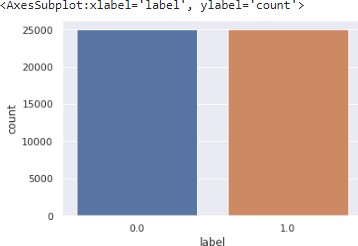
decoded = " ".join( [reverse\_index.get(i - 3, "#") for i in data[0]] )

# The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "start of sequence" and "unknown". print(decoded)

# this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert # is an amazing actor and now the same being director # father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film

#Adding sequence to data

# Vectorization is the process of converting textual data into numerical vectors and is a process that is usually applied once the text is cleaned. data = vectorize(data) label = np.array(label).astype("float32") labelDF=pd.DataFrame({'label':label}) sns.countplot(x='label', data=labelDF)



# Creating train and test data set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data,label, test\_size=0.20, random\_state=1)

X\_train.shape

(40000, 10000) X\_test.shape

(10000, 10000)

# Let's create sequential model from keras.utils import to\_categorical from keras import models from keras import layers model = models.Sequential()

# Input - Layer

# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 10,000 integers long.

# The input-layer takes 10,000 as input and outputs it with a shape of 50. model.add(layers.Dense(50, activation = "relu", input\_shape=(10000, ))) # Hidden - Layers

# Please note you should always use a dropout rate between 20% and 50%. # here in our case 0.3 means 30% dropout we are using dropout to prevent overfitting.

# By the way, if you want you can build a sentiment analysis without LSTMs, then you simply need to replace it by a flatten layer:

model.add(layers.Dropout(0.3, noise\_shape=None, seed=None)) model.add(layers.Dense(50, activation = "relu")) model.add(layers.Dropout(0.2, noise\_shape=None, seed=None)) model.add(layers.Dense(50, activation = "relu"))

# Output- Layer model.add(layers.Dense(1, activation = "sigmoid")) model.summary()

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |
| --- | --- | --- |
| dense (Dense) | (None, 50) | 500050 |
| dropout (Dropout) | (None, 50) | 0 |
| dense\_1 (Dense) | (None, 50) | 2550 |
| dropout\_1 (Dropout) | (None, 50) | 0 |
| dense\_2 (Dense) | (None, 50) | 2550 |
| dense\_3 (Dense) | (None, 1) | 51 |

=================================================================

Total params: 505,201

Trainable params: 505,201

Non-trainable params: 0

#For early stopping

# Stop training when a monitored metric has stopped improving.

# monitor: Quantity to be monitored.

# patience: Number of epochs with no improvement after which training will be stopped.

import tensorflow as tf

callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# We use the “adam” optimizer, an algorithm that changes the weights and biases during training.

# We also choose binary-crossentropy as loss (because we deal with binary classification) and accuracy as our evaluation metric.

model.compile(

optimizer = "adam", loss = "binary\_crossentropy", metrics = ["accuracy"]

)

from sklearn.model\_selection import train\_test\_split

results = model.fit( X\_train, y\_train, epochs= 2, batch\_size = 500, validation\_data = (X\_test, y\_test), callbacks=[callback]

)

# Let's check mean accuracy of our model print(np.mean(results.history["val\_accuracy"]))

# Evaluate the model score = model.evaluate(X\_test, y\_test, batch\_size=500) print('Test loss:', score[0]) print('Test accuracy:', score[1])

20/20 [==============================] - 1s 24ms/step - loss: 0.2511 - accuracy:

0.8986

Test loss: 0.25108325481414795

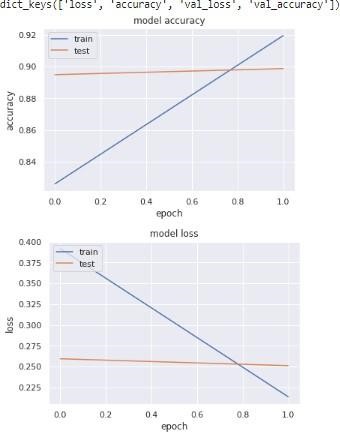
Test accuracy: 0.8985999822616577

#Let's plot training history of our model.

# list all data in history print(results.history.keys()) # summarize history for accuracy plt.plot(results.history['accuracy']) plt.plot(results.history['val\_accuracy']) plt.title('model accuracy') plt.ylabel('accuracy') plt.xlabel('epoch') plt.legend(['train', 'test'], loc='upper left') plt.show()

# summarize history for loss plt.plot(results.history['loss']) plt.plot(results.history['val\_loss']) plt.title('model loss') plt.ylabel('loss')

plt.xlabel('epoch') plt.legend(['train', 'test'], loc='upper left') plt.show()



**Conclusion**- In this way we can Classify the Movie Reviews by using DNN.

# Assignment Question

1. What is Binary Classification?
2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle?
5. What is Adam Optimizer?